Leonid Kalinichenko
Rainer Manthey
Bernhard Thalheim
Uwe Wloka (Eds.)

LNCS 2798

Advances in Databases and Information Systems

7th East European Conference, ADBIS 2003
Dresden, Germany, September 2003
Proceedings

Springer

Lecture Notes in Computer Science 2798
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Leonid Kalinichenko   Rainer Manthey
Bernhard Thalheim   Uwe Wloka (Eds.)

Advances in Databases and Information Systems

7th East European Conference, ADBIS 2003
Dresden, Germany, September 3-6, 2003
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Leonid Kalinichenko
Russian Academy of Science, Institute of Informatics Problems
Vavilova 44-2, Moscow, 117333, Russia
E-mail: leonidk@synth.ipi.ac.ru

Rainer Manthey
University of Bonn, Department of Computer Science III
Römerstr. 164, 53117 Bonn, Germany
E-mail: manthey@cs.uni-bonn.de

Bernhard Thalheim
Brandenburg University of Technology, Institute of Computer Science
Postfach 101344, 03013 Cottbus, Germany
E-mail: thalheim@informatik.tu-cottbus.de

Uwe Wloka
University of Applied Sciences, Department of Information Technology/Mathematics
Friedrich-List-Platz 1, 01069 Dresden, Germany
E-mail: wloka@informatik.htw-dresden.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.2, H.3, H.4, H.5, J.1

ISSN 0302-9743
ISBN 3-540-20047-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10931912 06/3142 5 4 3 2 1 0

Preface

This volume contains 29 submitted and 2 invited papers presented at the tenth East-European Conference on Advances in Databases and Information Systems (ADVIS 2003), which took place in Dresden, Germany, September 3–6, 2003.

An international program committee of 42 members from 24 countries selected these contributions from 86 submissions. Eight additional contributions were selected as short papers and have been published in a separate volume of local proceedings by the organizing institution. For the first time, ADBIS also included an industrial program consisting of nine submitted presentations by representatives of commercial companies active in the database market.

ADBIS 2003 was the tenth scientific event taking place under the acronym ADBIS, and thus marks a first "jubilee" for this young, but by now well-established, series of conferences. ADBIS was founded by the Moscow ACM SIGMOD Chapter in 1993. In 1994–1996 ADBIS was held in the form of workshops organized by the MOSCOW ACM SIGMOD Chapter in collaboration with the Russian Foundation for Basic Research. In this period, a number of international guests were invited to Moscow every year in order to improve and consolidate contacts between the national research community in Russia and the research community worldwide. International program committees for ADBIS were established in 1995, contributing to the quality of the selection process and thus of the conference contributions in general. In 1996, following discussions with Dr. Won Kim (ACM SIGMOD Chair at that time) ADBIS was extended into a regular East European conference to provide a forum for promoting interaction and collaboration between the database research communities of Eastern and Central Europe and the rest of the world. The Moscow ACM SIGMOD Chapter continues to serve as a focal point of the series.

In 1997 the ADBIS conferences started their "tour of Eastern and Central Europe", beginning in St. Petersburg (Russia, 1997), and continuing in Poznan (Poland, 1998), Maribor (Slovenia, 1999), Prague (Czech Republic, as a joint ADBIS-DASFAA conference in 2000), Vilnius (Lithuania, 2002), Bratislava (Slovakia, 2002), and in Dresden (Germany, 2003). The next ADBIS conference is to take place in Budapest (Hungary, 2004). Since 1998, ADBIS proceedings have been published in Springer-Verlag's LNCS series (full papers) and in local volumes (short papers) and have been included in the ACM SIGMOD Anthology, too. The ADBIS Steering Committee currently consists of representatives from Austria, Bulgaria, Czech Republic, Greece, Estonia, Germany, Hungary, Latvia, Lithuania, Poland, Russia, Romania, Serbia and Montenegro, Slovakia, Slovenia, and the Ukraine.

Our special thanks go to the many contributors, the reviewers and program committee members, all the members of the organizing committee, and last but not least to Springer-Verlag for their efficient and friendly support in publishing the proceedings in their renowned LNCS series.

July 2003

Leonid Kalinichenko
Rainer Manthey
Bernard Thalheim
Uwe Wloka

Organization

The Seventh East-European Conference on Advances in Databases and Information Systems (ADBIS) was organized by members of the Department of Information Technology/Mathematics of the Hochschule für Technik und Wissenschaften (University of Applied Sciences) in Dresden and of the Department of Computer Science of the Brandenburg University of Technology in Cottbus, Germany, in cooperation with the Moscow ACM SIGMOD Chapter.

General Chair

Bernhard Thalheim (Brandenburg University of Technology
in Cottbus, Germany)

Program Committee Co-chairs

Leonid Kalinichenko (Russian Academy of Science, Moscow, Russia)
Rainer Manthey (University of Bonn, Germany)

Program Committee Members

Leopoldo Bertossi (Carleton University, Ottawa, Canada)
Stéphane Bressan (National University of Singapur)
Albertas Caplinskas (Institute of Mathematics and Informatics,
Vilnius, Lithuania)
Bogdan Czejdo (Lodola University, New Orleans, USA)
Johann Eder (University of Klagenfurt, Austria)
Piero Fraternali (Politecnico di Milano, Italy)
Janis Grundspenkis (Riga Technical University, Latvia)
Remigijus Gustas (Karlstad University, Sweden)
Theo Härder (University of Kaiserslautern, Germany)
Hele-Mai Haav (Talinn Technical University, Estonia)
Tomas Hruska (Brno University of Technology, Czech Republic)
Christian Jensen (Aalborg University, Denmark)
Hannu Kangassalo (University of Tampere, Finland)
Mikhail Kogalovsky (Russian Academy of Sciences, Russia)
Tok Wang Ling (National University of Singapore)
Peri Loucopoulos (University of Manchester, United Kingdom)
Yannis Manolopoulos (Aristotle University of Thessaloniki, Greece)
Mikhail Matskin (Royal Institute of Technology, Stockhol, Sweden)
Klaus Meyer-Wegener (University of Erlangen-Nürnberg, Germany)
Robert Meersman (Free University of Brussels, Belgium)
Tadeusz Morzy (Poznan University of Technology, Poland)

Pavol Navrat (Slovak University of Technology, Bratislava, Slovakia)
Nikolay Nikitchenko (National Taras Shevchenko University of Kiev, Ukrainia)
Kjetil Norvag (Norwegian Univ. of Science and Technology, Trondheim, Norway)
Boris Novikov (University of St. Petersburg, Russia)
Oscar Pastor (Valencia University of Technology, Spain)
Jacek Plodzien (Polish Academy of Sciences, Warszawa, Poland)
Jaroslav Pokorny (Charles University, Prague, Czech Republic)
Boris Rachev (Technical University of Varna, Bulgaria)
Marek Rusinkiewicz (Telcordia Research, Morristown, USA)
Gunter Saake (University of Magdeburg, Germany)
Silvio Salza (Università di Roma “La Sapienza”, Roma, Italy)
George Samaras (University of Cyprus, Nicosia, Cyprus)
Joachim W. Schmidt (Technical University of Hamburg-Harburg, Germany)
Heinz Schweppe (Technical University of Berlin, Germany)
Stefano Spaccapietra (École Polytechnique Fédérale de Lausanne, Switzerland)
Julius Stuller (Czech Academy of Sciences, Czech Republic)
Toni Urpí (Technical University of Barcelona, Spain)
Tatjana Welzer (University of Maribor, Slovenia)
Robert Wrembel (Poznan University of Technology, Poland)
Vladimir Zadorozhny (University of Pittsburgh, USA)
Alexander Zamulin (Russian Academy of Sciences, Russia)

Additional Reviewers

Albert Abelló
Sven Van Acker
Fabrizio d’Amore
Andreas Behrend
Marco Brambilla
Dmitry Buy
Diego Calvanese
Ting Chen
Sara Comai
Marjan Druzovec
Antonio Corral
Liangliang Fang
Miguel Garcia Gutierrez
Hagen Höpfner
Andrzej Jaszkiewicz
Viviane Jonckers
Tomas Karban
Kyriakos Karenos
Dimitrios Katsaros
Evangelia Kavakli
Kristiina Kindel
Marite Kirikova
Jan Kniat
Christian Koncilia
Zbyszko Krolikowski
Marek Lehmann
Wai Lup Low
Maristella Matera
Alexandros Nanopoulos
Christoforos Panagiotou
Stavros Papastavrou
Ekaterina Pavlova
Horst Pichler
Nikos Prekas
Francesco Quaglia
Kai-Uwe Sattler
Eike Schallehn
Hans-Werner Sehring
Chara Skoutelli
Cezary Sobaniec
Jerzy Stefanowski
Michael Vassilakopoulos
Panos Vassiliadis

Organizing Committee

Chairman: Uwe Wloka (University of Applied Sciences Dresden)

Gunter Gräfe, Sieglinde Grosch, Elke Hamel, Axel Toll, Jutta Walther
(University of Applied Sciences Dresden)
Katrin Fleischer, Hartmut Fussan
(ZAFT at University of Applied Sciences Dresden)
Steffen Jurk, Karla Kersten, Thomas Kobienia, Bernhard Thalheim
(Brandenburg University of Technology Cottbus)

ADBIS Steering Committee

Chairman: Leonid Kalinichenko (Russia)

Andras Benczur (Hungary)	Radu Bercaru (Romania)
Albertas Caplinskas (Lithuania)	Johann Eder (Austria)
Janis Eiduks (Latvia)	Hele-Mai Haav (Estonia)
Mirjana Ivanovic (Serbia and Montenegro)	Mikhail Kogalovsky (Russia)
Yannis Manolopoulos (Greece)	Rainer Manthey (Germany)
Tadeusz Morzy (Poland)	Pavol Návrat (Slovakia)
Boris Novikov (Russia)	Jaroslav Pokorny (Czech Republic)
Boris Rachev (Bulgaria)	Anatoly Stogny (Ukraine)
Tatjana Welzer (Slovenia)	Viacheslav Wolfengagen (Russia)

Table of Contents

Semantic Web Services: The Future of Integration!

Christoph Bussler

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway, Ireland
Chris.Bussler@DERI.ie

1 Semantic Integration

The hype around current Web Service technology and standards proposals is mind-boggling. Promises are made about the ease, speed, reliability and persuasiveness of Web Service technology all centered on the three basic standard proposals SOAP [12], WSDL [15] and UDDI [13]. Web Service technology is praised as the integration technology of tomorrow. Addressing the integration space, more proposals are stacked onto the three basic ones covering composition like BPEL4WS [1], ebXML [7], BPML [2], transaction support like WS-C [14] and WS-T [16], security, and many, many more creating the unavoidable future Web Service technology jungle [4].

The main dominant issue when applying integration technology [3], however, is not addressed by any of the Web Service proposals at all: semantics. Without a solution for the semantic aspect of integration, none of the new integration technologies centered on Web Services will improve integration solutions; just provide a different implementation technology, not a 'better' one at all. Not focusing on semantics is a recipe for failure: enabling integration will be as hard as ever since ensuring the semantic compatibility of the integrated entities remains manual and hence cumbersome and error-prone work.

Integration requires semantics in two areas:

- **Events.** Events are the messages that are sent between integrated entities (e.g., businesses or application systems) that carry intent. For example, sending a purchase order is meaningless unless the intent to create, update, inquire or delete it is part of the transmission. This distinguishes a message from an event. Events are directed and have intent besides the business data. Furthermore, different integrated entities follow different ontologies [8] and therefore have different data type models. When events are sent from one entity to another one, events have to be transformed structurally as well as semantically [3].
- **Behavior.** Events are in most cases never sent in isolation. Events follow communication patterns between the integrated entities. The reason for this is that events are dependent on each other like acknowledgements or changes to previous intents. The communication pattern has to be flawless in order to avoid compromising the internal state of the integrated entities (causing business disruption). It is therefore essential to describe communication patterns (sometimes called 'public processes') semantically in such a way that the meaning of communication is completely captured [9]. This avoids event exchange failures.

The term ‘Semantic Web Services’ was coined in order to reflect the importance of the semantic aspect of integration. Semantic Web Services address the semantic description of events and behavior as their core concepts. Additional supporting concepts are included like event transformation and entity management to provide a complete semantic integration model.

Semantic Web Services is the combination of Semantic Web technology with Web Service technology together with additional essential concepts required to make the semantic integration work. Semantic Web Services use Semantic Web technology in order to allow semantically correct and reliable integration. They are based on Web Service technology in order to benefit from the work in this field.

Several efforts are taking place currently world-wide in order to work on and advance the concepts of Semantic Web Services. These are the newly founded Digital Enterprise Research Institute (DERI) [6], a European project called Semantic Web-enabled Web Services (SWWS) [10] and a standards initiative call Semantic Web Service Initiative (SWSI) [11] with links into the DAML-S [5] program.

All these various efforts realized that the future of integration (and the Semantic Web for that matter) has to tackle the semantics in context of integration in form of Semantic Web Services.

References

[1] BPEL4WS. Business Process Execution Language for Web Services, Version 1.1. www.106.ibm.com/developerworks/webservices/library/ws-bpel/

[2] BPML. Business Process Modeling Language. www.bpml.org

[3] Bussler, C.: B2B Integration. Springer, Berlin Heidelberg New York, 2003

[4] Cover, R.: Cover Pages. www.oasis-open.org/cover/sgml-xml.html

[5] DAML-S. DAML Services. www.daml.org/services/

[6] Digital Enterprise Research Institute (DERI). deri.semanticweb.org

[7] ebXML. ebXML Business Process Specification Schema Version (BPSS) 1.01. www.ebxml.org/specs/index.htm

[8] Fensel, D.: Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, Berlin Heidelberg New York, 2001

[9] Fensel, D.; Bussler, C.: The Web Service Modeling Framework WSMF. In: Electronic Commerce Research and Applications, Vol. 1, Issue 2, Elsevier Science B.V., Summer 2002

[10] Semantic Web-enabled Web Services (SWWS). swws.semanticweb.org

[11] Semantic Web Service Initiative (SWSI). www.swsi.org

[12] SOAP. Simple Object Access Protocol (SOAP) 1.2. www.w3.org/TR/soap12-part1/ and www.w3.org/TR/soap12-part2/

[13] UDDI. Universal Description, Discovery and Integration. www.uddi.org

[14] WS-C. Web Services Coordination (WS-Coordination). www.ibm.com/developerworks/library/ws-coor/

[15] WSDL. Web Service Description Language (WSDL) 1.1. www.w3.org/TR/wsdl

[16] WS-T. Web Services Transaction (WS-Transaction). www.ibm.com/developerworks/webservices/library/ws-transpec/

Bioinformatics Databases: State of the Art and Research Perspectives

François Bry and Peer Kröger

Institute for Computer Science
University of Munich, Germany
{bry,kroegerp}@informatik.uni-muenchen.de

Abstract. Bioinformatics or computational biology, i.e. the application of mathematical and computer science methods to solving problems in molecular biology that require large scale data, computation, and analysis, is a research area currently receiving a considerable attention. Databases play an essential role in molecular biology and consequently in bioinformatics. molecular biology data are often relatively cheap to produce, leading to a proliferation of databases: the number of bioinformatics databases accessible worldwide probably lies between 500 and 1.000. Not only molecular biology data, but also molecular biology literature and literature references are stored in databases. Bioinformatics databases are often very large (e.g. the sequence database GenBank contains more than 4×10^6 nucleotide sequences) and in general grows rapidly (e.g. about 8000 abstracts are added every month to the literature database PubMed). Bioinformatics databases are heterogeneous in their data, in their data modeling paradigms, in their management systems, and in the data analysis tools they supports. Furthermore, bioinformatics databases are often implemented, queried, updated, and managed using methods rarely applied for other databases. This presentation aims at introducing in current bioinformatics databases, stressing their aspects departing from conventional databases. A more detailed survey can be found in [1] upon which this presentation is built on.

References

[1] François Bry and Peer Kröger. *A Computational Biology Database Digest: Data, Data Analysis, and Data Management*. Distributed and Parallel Databases, Vol. 13, pp. 7–42, Kluwer Academic Press, 2003.

Multirepresentation in Ontologies

Djamal Benslimane[1], Christelle Vangenot[2], Catherine Roussey[1], and Ahmed Arara[1]

[1] LIRIS, Université Lyon 1, IUT A 69622 Villeurbanne, France
{djamal.benslimane,catherine.roussey,ahmed.arara}@iuta.univ-lyon1.fr
[2] Swiss Federal Institute of Technology
Database Laboratory, Lausanne, Switzerland
christelle.vangenot@epfl.ch

Abstract. The objective of this paper is to define an ontology language to support multiple representations of ontologies. In our research, we focus on the logic-based ontology languages. As a matter of fact, we will consider only languages that are based on description logics (DLs). At first, we propose a sub-language of DL as an ontology language. Furthermore we achieve multiple representations of ontological concepts by extending such sub-language through the use of stamping mechanism proposed in the context of multiple representation of spatial databases. The proposed language should offer a modest solution to the problem of multirepresentation ontologies.

1 Introduction

1.1 Motivation

Ontologies, as shared, common, representational vocabulary play a key role in many artificial intelligence applications and information integration. They are used as the basis for communication among multiple, heterogeneous systems. Ontology representation is an important issue that has been investigated and several languages are commonly used such as Ontolingua [10], KARL [3], OIL [11], OCML [15].

There is no natural unique way to look at data and there is no natural unique way to represent it. While the real world is supposed to be unique, its representation depends on the intended purpose: every representation of reality conveys a user-specific representation of the real world. Thus, different applications that share interest in the same real-word phenomena may have different perceptions and therefore require different representations. Differences may arise in all facets that make up a representation: what amount of information is kept, how it is described, how it is organized (in terms of data structures), how it is coded, what constraints, processes and rules apply, how it is presented, what are the associated spatial and temporal frameworks, etc.

Most frequently, work on ontologies aims at developing a single-world ontology, i.e. an ontology that represents a given conceptualization of the real world from a given perspective. However, many researchers recognize that semantics is context-dependent: there is no single truth, and interpretation of concepts in particular depends on the context in which the concepts are used. A non-normative ontology should provide contextual definitions and contextual data structures to represent the diversity of perceptions and focuses. One example of interests where context is needed is in semantic interoperability. Ontologies play an essential role in semantic interoperation of information systems. Unfortunately, semantic interoperability is not easy to achieve because related knowledge is most likely to be described in different terms, using different assumptions and different data structures. Multi-representation ontologies can enable this integration. For instance, [9] introduces the concept of role to link the same concept described in several ontologies used to integrate information systems. Moreover, context may be used to define user profiles in ontologies and thus allows to select a subset of the ontology. For instance, in [13], context allows to filter results of multimedia database querying according to user profiles.

Domain ontologies are constructed by capturing a set of concepts and their links according to a given context. A context can be viewed as various criteria such as the abstraction paradigm, the granularity scale, interest of user communities, and the perception of ontology developer. So, the same domain can have more than one ontology, where each one of them is described in a particular context. We call each one of these ontologies a **MonoRepresentation ontology (MoRO)**. Thus, concepts in MoRO are defined with one and only one representation. Our motivation is to see how we can describe ontology according to several contexts at the same time. We shall call such ontology a **MultiRepresentation Ontologies (MuRO)**. A MuRO is an ontology that characterizes an ontological concept by a variable set of properties or attributes in several contexts. So, in a MuRO, a concept is defined once with several representations, such that a single representation is available for one context.

The multi-representation problem is commonly known in the discipline of spatial databases and object-oriented modeling. Spaccapiatra et al. [19] have investigated the problem in spatial databases to associate different geometry, resolution and scale with the same spatial object. In their work, they proposed an extension of existing ER-based model called MADS [17]. A stamping mechanism of data elements (concepts, attributes, instances) and relationships is suggested to enable manipulations of data elements from several representations. In object-oriented modeling, several object-role languages have been proposed to represent an object by a single structure which is augmented dynamically by specifc information (roles) related to its different facets [18,14].

Existing ontology languages are not capable of defining a single MuRO ontology. For instance, there isn't any possibility to define a unique road concept with different sets of properties at the same time. So, our motivation is to deal with the problem of multirepresentation in the context of ontology languages. In our research, we focus on the logic-based ontology languages. As a matter of fact, we will consider only languages that are based on description logics (DLs). DLs are a subset of first order logic used to describe knowledge in terms of concepts and roles to automatically derive classification taxonomies and provide reasoning services. Concepts in DL are intentionally described to specify the properties that individuals must satisfy in order for them to belong to the concept. DL-based ontology languages namely OWL [22], OIL, and DAML+OIL [21] are gaining support from many organisations and technical institutions because of their importance in data, information, and knowledge interoperation and exchange.

1.2 Contribution and Organisation of the Paper

Our objective is to define a contextual ontology language to support multiple representations of ontologies. The underlying key idea of our work is to adapt the stamping mechanism proposed in [19] to serve the needs of multirepresentation in spatial databases to our needs of ontology language requirements. To achieve this goal, we first propose a sub-language of DL as an ontology language and then we extend it by introducing stamping mechanism to the constructs of the DL language to allow multiple representations of concepts. The remainder of the paper is organised as follows. Section 2 presents the sub-language of DL that we propose as ontology language. Section 3 briefly summarises the multi-representation paradigm based on stamping mechanism. Section 4 is devoted to the presentation of some extension of DL to support multirepresentation needs. Section 5 presents some related work and finally, section 6 will conclude the paper.

2 Multirepresentation Paradigm Based on Stamping Mechanism

A stamping mechanism to characterise database elements was proposed in [20] to support multiple representations of the same data. Stamps, used to characterise modeling contexts defined in a database, have a twofold semantic: they allow to distinguish the multiple representations of the same phenomenon and also to filter access to data during querying. Hence, a concept can be used in one or more context and each representation of a concept is stamped or labeled differently.

This proposal can be illustrated through use of an example. Let us consider two representations of the real world, identified by the stamps s_1 and s_2, corresponding to *Road traffic* and *Driving school* contexts.

The description of the stamped concept $Vehicle$ is as it follows:

Type $Vehicle$ **(**s_1**,**s_2**)**
s_1: Speed (1,1) : number,
s_1: VehicleType (1,1) : string,
s_1, s_2: LicencePlate (1,1) : string,
s_1, s_2: AssurancePolicy (1,1) : string,
s_1: RegistrationDate (1,1) : date,
s_2: RegistrationDate (1,1) : string,
s_1: Driver (1,1) : string,
s_2 : CarModel (1,1) : string

Stamps are written as arguments after type name of concept (i.e. type $Vehicle$ (s_1, s_2). A list of stamps is written before each atribute name. Stamping attributes allows for the definition of personalized concepts whose definition is varying according to stamps. Thus, in the context identified by s_1, the Vehicle concept is described by the attributes *Speed*, *VehicleType*, *LicencePlate*, *AssurancePolicy*, *RegistrationDate* (of domain Date) and *Driver*. In the context identified by s_2, it is described by the attributes *LicencePlate*, *AssurancePolicy* (as in context s_1), *RegistrationDate* but of a different attribute domain (string) and *CarModel*.

Stamps apply at the schema level (meta-data) thus allowing attributes to have:

– a single definition visible in a particular context, like for instance the attribute Speed,
– several definitions, i.e. different cardinalities or different domains of values according to the context, like the attribute *RegistrationDate*.

Stamps also apply at the data level and thus allow to partition the population of concepts into subpopulations having different stamps. It is useful when some instances of a multirepresentation concept are not shared. Stamping an instance allows to specify by which user an instance may be accessed: If instances corresponding to motor vehicles are stamped only with stamp s_2 and all other vehicles with stamps s_1 and s_2, users of the Driving school database have access only to motor vehicles. Like concepts, links that connect concepts can be stamped with the same semantics and according to the same rules.

The stamping technique is particularly of interest to deal with multiple representations of data as it allows:

– to define a representation context with a stamp that is a tuple of values corresponding to any number of meta-data items (viewpoint, resolution, time).
– to associate stamps with concepts, links, attributes, instances. Thus, the constructors of our model are orthogonal.

3 Adding Multirepresentation Mechanism to Description Logics

This section presents an extension of the description logics fragment given in appendix. This extension is able to describe the context varying aspect of concepts. Such concepts will be called contextual concepts, i.e. concepts with more than one representation, each representation is available in a given context.

3.1 Contextual Constructors

In DL, we have the notions of concepts (that we denote classical concepts for more clarity) as a set of individuals (unary predicates) and roles (binary predicates) as attributes or relationships. Complex concepts are derived from atomic concepts by applying DL constructors (also will be denoted as classical constructors). For our requirements of multi-representation ontologies, we propose the notion of contextual concepts that may describe concepts associated with different contexts.

Contextual concepts are derived basically from atomic concepts by using a set of classical and/or contextual constructors. Contextual constructors are defined by specializing the classical ones to allow the construction of a concept that is partially or completly available in some contexts.

For the needs of multirepresentation, we add the following rule to the syntax defined in classical description logics language.

Definition 1. (Syntax of contextual concept terms) *Let $s_1, \cdots, s_m$ be a set of context names. Contextual concept terms C and D can be formed by means of the following syntax:*

$$
\begin{array}{ll}
C, D \longrightarrow & \\
\quad \exists_{s_1,\cdots,s_m} R.C \mid & \text{(contextual existential quantification)} \\
\quad \forall_{s_1,\cdots,s_m} R.C \mid & \text{(contextual value restriction)} \\
\quad (\leq_{s_1,\cdots,s_m} nR) \mid & \text{(contextual at most number restriction)} \\
\quad (\geq_{s_1,\cdots,s_m} nR) \mid & \text{(contextual at least number restriction)} \\
\quad C \sqcap_{s_1,\cdots,s_m} D & \text{(contextual conjunction)}
\end{array}
$$

The definition of non-contextual concepts remains always possible. Such concepts will exist in all contexts with single representation.

The semantics of the non-contextual language is extended with the contextual notions. To define the semantics of the contextual constructors, we assume having a set of context names $S = \{s_1, s_2, ..., s_t\}$.

Definition 2. (Semantics of contextual concept terms) *The semantics of the contextual part of the language is given by a contextual interpretation defined in a context j over $\mathcal{S}$. A context j is either a simple context name belonging to $\mathcal{S}$ or a composed context defined as a conjunction of simple contexts*[1]*. A contextual interpretation $\mathcal{CI} = (\mathcal{I}_0, \mathcal{I}_1, \cdots, \mathcal{I}_j, \cdots, \mathcal{I}_t)$ is obtained by associating to each context j a non-contextual interpretation $\mathcal{I}_j = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}^j})$, which consists of an interpretation domain $\Delta^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}^j}$. The interpretation function $\cdot^{\mathcal{I}^j}$ maps each atomic concept $A \in \mathcal{C}$ to a subset $A^{\mathcal{I}}j \subseteq \Delta^{\mathcal{I}}$ and each role name $R \in \mathcal{R}$ to a subset $R^{\mathcal{I}}j \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Let $Co(j)$ be a function which returns a set of all context names appearing in a simple/composed argument context j*[2]*. The extension of $\cdot^{\mathcal{I}^j}$ to arbitrary concepts is inductively defined as follows:*

[1] $j = s_1$ and $j = s_1 \wedge s_2$ are two examples of simple context and composed context respectively.

[2] $Co(s_1 \wedge s_2) = \{s_1, s_2\}$ and $Co(s_1) = \{s_1\}$ are two examples of the function Co.

$(\exists_{s_1,\cdots,s_m} R.C)^{\mathcal{I}}j = \{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in R^{\mathcal{I}}j \wedge y \in C^{\mathcal{I}}j \wedge Co(j) \cap \{s_1, \cdots, s_m\} \neq \emptyset\}$
$(\forall_{s_1,\cdots,s_m} R.C)^{\mathcal{I}}j = \{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in R^{\mathcal{I}}j \rightarrow y \in C^{\mathcal{I}}j \wedge Co(j) \cap \{s_1, \cdots, s_m\} \neq \emptyset\}$
$(\leq_{s_1,\cdots,s_m} nR)^{\mathcal{I}}j = \{x \in \Delta^{\mathcal{I}} \mid Co(j) \cap \{s_1, \cdots, s_m\} \neq \emptyset \wedge \quad \|\{y \mid (x,y) \in R^{\mathcal{I}}j\}\| \leq n\}$
$(\geq_{s_1,\cdots,s_m} nR)^{\mathcal{I}}j = \{x \in \Delta^{\mathcal{I}} \mid Co(j) \cap \{s_1, \cdots, s_m\} \neq \emptyset \wedge \quad \|\{y \mid (x,y) \in R^{\mathcal{I}}j\}\| \geq n\}$
$(C \sqcap_{s_1,\cdots,s_m} D)^{\mathcal{I}}j = \{x \in \Delta^{\mathcal{I}} \mid x \in C^{\mathcal{I}}j\} if Co(j) \cap \{s_1, \cdots, s_m\} = \emptyset,$
$= \{x \in \Delta^{\mathcal{I}} \mid x \in C^{\mathcal{I}}j \cap D^{\mathcal{I}}j\} if Co(j) \cap \{s_1, \cdots, s_m\} \neq \emptyset$

In the following, we briefly describe contextual constructors accompanied by simple examples.

The contextual value restriction constructor ($\forall_{s_1,\cdots,s_m} R.C$): It will define a new concept all of whose instances are related via the role R only to the individuals of class C and in the contexts s_1 to s_m. For example, in two contexts s_1 and s_2, the concept Employee is defined as individuals with either an attribute EmployeeNumber in context s_1 or an attribute LastName in context s_2. The concept is expressed as it follows:

$$\text{Employee} = \forall_{s_1} EmployeeNumber.Number \sqcup \forall_{s_2} LastName.String$$

Outside of the two contexts s_1 and s_2, the concept *Employee* corresponds to an empty set.

The contextual existential quantification constructor ($\exists_{s_1,\cdots,s_m} R.C$) will construct a new concept all of whose instances are related via the role R to at least one individual of type C and only in contexts s_1 to s_m. For example, the following expression describes that student is an individual that has at least one graduation diploma in context s_1 and in the same time participates in at least one course in context s_2:

$$\text{Student}= \exists_{s_1} Diploma.Graduate \sqcap \exists_{s_2} Register.Course$$

It should be noted that the interpretation of the expression Student in the context s_1 or s_2 separately will give us an empty concept. On the contrary, the interpretation of Student in the context $(s_1 \wedge s_2)$ will correspond to some individuals satisfying the two conditions of the expression.

The contextual number (at most, at least) restriction constructors ($\leq_{s_1,\cdots,s_m} nR, \geq_{s_1,\cdots,s_m} nR$): They specify the number of role-fillers. The $\leq_{s_1,\cdots,s_m} nR$ is used to indicate the maximum cardinality whereas, the expression $\geq_{s_1,\cdots,s_m} nR$ indicates the minimum cardinality in the given contexts s_1 to s_m. The following example illustrates two cardinalities in different contexts: context s_1 where a man is allowed for at most one wife at one period of time, and context s_2 where he is allowed to have at most four wives at the same time. In the two contexts s_1 and s_2, the type of $NumberOfWife$ is $Number$.

$$\text{Man}= (\leq_{s_1} 1 Number0fWife \sqcup \leq_{s_2} 4 Number0fWife) \sqcap$$
$$\forall_{s_1,s_2} NumberOfWife.Number$$

The contextual conjunction ($C \sqcap_{s_1,\cdots,s_m} D$): It will define either a concept resulting of the conjunction of the two concepts C and D in the defined contexts s_1 to s_m, or a concept equivalent to concept C outside of all the given contexts (s_1 to s_m). For example, the following expression decsribes a Manager as being a person of sex female and who either has at least one responsability in the context s_1 or manages at leat one project in the context s_2. Outside of the two contexts s_1 and s_2, Manager is only defined as being a person of sex female and nothing else.

$$Manager = (Person \sqcap \forall Sex.Female) \sqcap_{s_1,s_2}$$
$$(\exists_{s_1} Responsability.String \sqcup \exists_{s_2} Manage.Project)$$

A terminological axiom $A = D$ is satisfied by a contextual interpretation $\mathcal{CI}$ if $A^{\mathcal{I}}j = D^{\mathcal{I}}j$ for every context j. A contextual interpretation $\mathcal{CI}$ is a model for a *TBox* $\mathcal{T}$ if and only if $\mathcal{CI}$ satisfies all the assertions in $\mathcal{T}$. Finally, a concept D subsumes a concept C iff $C \sqsubseteq D$ is satisfied in every contextual interpretation $\mathcal{CI}$. The contextual constructors proposed in this paper can be viewed as an adaptive constructors that specialize the non-contextual ones. Thus, the subsumption for the language will be decidable since it constitutes a subset of the description logic $\mathcal{ALCNR}$ [7].

3.2 Example of Multirepresentation Ontology

Table 1 presents a very simple multirepresentation ontology described in our language. Two contexts are considered: the traffic control and driving school designated by s_1 and s_2 respectively.

4 Related Work and Discussion

In the following, a brief discussion of how this paper is related to other approaches that deal with the issue of modeling and reasoning about conceptual schemas in various contexts.

Modal Description Logics. The classical description logics are not intended to deal with the dynamic aspects of knowledge representation such as spatial, temporal, beliefs, obligations, etc [23]. Modal descriptions logics, however, are adequate to represent such notions. Modal description logics can be defined as description logics (DLs) augmented by some modal operators like (a) necessity to express that a concept holds in some given worlds (i.e. all worlds in the future), (b) possibility to express the existence of a world in which a concept holds. Different modal description logics have been proposed to deal with this dynamic aspect of concepts. As a matter of fact, we found out that temporal logics are the most suitable and closely related to our approach. In temporal description logics, some temporal operators (for example Until and Since) are introduced to define a concept in the past, present and future tense [4]. In a similar manner, we can consider the tense in general (past, present and future) as a characterisation of a context. According to such characterizing criteria, the temporal description logics give us the evolution feature of the whole concept from

Table 1. Multirepresentation Ontology Example for Road Traffic and Driving School Contexts

Concepts defintion
$\mathbf{Road} = (\leq_{s_1} 1StartPoint) \sqcap (\geq_{s_1} 1StartPoint) \sqcap (\forall_{s_1} StartPoint.Point) \sqcap (\leq_{s_1} 1EndPoint) \sqcap (\geq_{s_1} 1EndPoint) \sqcap (\forall_{s_1} EndPoint.Point) \sqcap (\leq_{s_1,s_2} 1Name) \sqcap (\geq_{s_1,s_2} 1Name) \sqcap (\forall_{s_1,s_2} Name.String) \sqcap (\leq 1With_{s_1}) \sqcap (\geq_{s_1} 1With) \sqcap (\forall_{s_1} With.Number) \sqcap (\leq_{s_1} 1Lanes) \sqcap (\geq_{s_1} nLanes) \sqcap (\forall_{s_1} Lanes.Number) \sqcap (\leq_{s_1} 1SpeedLimitWith) \sqcap (\geq_{s_1} 2SpeedLimit) \sqcap (\forall_{s_1} SpeedLimit.Number) \sqcap (\leq_{s_2} 1RoadType) \sqcap (\geq_{s_2} 1RoadType) \sqcap (\forall_{s_2} RoadType.String)$
$\mathbf{Vehicle} = (\leq_{s_1} 1Speed) \sqcap (\geq_{s_1} 1Speed) \sqcap (\forall_{s_1} Speed.Number) \sqcap (\leq_{s_1} 1VehicleType) \sqcap (\geq_{s_1} 1VehicleType) \sqcap (\forall_{s_1} VehicleType.String) \sqcap (\leq_{s_1,s_2} 1LicencePlate) \sqcap (\geq_{s_1,s_2} 1LicensePlate) \sqcap (\forall_{s_1,s_2} LicensePlate.String) \sqcap (\leq_{s_1,s_2} 1InsurancePolicy) \sqcap (\geq_{s_1,s_2} 1InsurancePolicy) \sqcap (\forall_{s_1,s_2} InsurancePolicy.String) \sqcap (\leq_{s_2} 1CarModel) \sqcap (\geq_{s_2} 1CarModel) \sqcap (\forall_{s_2} CarModel.String)$
$\mathbf{RoadSign} = (\leq_{s_1} 1Type) \sqcap (\geq_{s_1} 1Type) \sqcap (\forall_{s_1} Type.string) \sqcap (\leq_{s_1} 1Location) \sqcap (\geq_{s_1} 1Location) \sqcap (\forall_{s_1} Location.String) \sqcap (\leq_{s_1} 1Purpose) \sqcap (\geq_{s_1} nPurpose) \sqcap (\forall_{s_1} Purpose.String)$
$\mathbf{Student} = (\leq_{s_2} 1Name) \sqcap (\geq_{s_2} 1Name) \sqcap (\forall_{s_2} Name.String) \sqcap (\leq_{s_2} 1Birthdate) \sqcap (\geq_{s_2} 1Birthdate) \sqcap (\forall_{s_2} Birthdate.Date)$
$\mathbf{Instructor} = (\leq_{s_2} 1Name) \sqcap (\geq_{s_2} 1Name) \sqcap (\forall_{s_2} Name.String)$
...

one context to another. Hence, this work is different from ours in the sense that it does not give us the ability to explicitly designate context names of contexts as in the temporal case where a precise time is given.

RDF Context. Resource Description Framework (RDF) is an emerging standard for the representation and exchange of metadata in the semantic web. RDF and RDF schema (RDFS) have been exploited in a variety of applications only to name few: resource discovery, cataloguing , digital libraries,etc. Recently, RDF(S) has become a good candidate for ontological knowledge representation. RDFS, as an ontology language, however, suffers from the power of expressivity and automated reasoning. A. Delteil and C. Faron-Zucker have proposed in [8] an extension of RDFS to go beyond the existing triple RDF statement-that is: the resource (subject), the property (predicate) and the value (object). They suggested a Defined Resource Description Framework (DRDFS) that enables the definition of class, property, and axiom definition to express contextual knowledge on the Web. The approach is based on the conceptual graph (CG) features. In this work, contextual annotations are proposed using extended RDF primitives namely *context*, *isContextOf*, and *referent*. The problem with this approach is that: it considers a context as a whole cluster of RDF statements (annotations). Moreover, rules based on the translation of CG graphs are needed to construct RDF contexts. In our approach, we consider a concept that may coexist in several contexts but with variable sets of properties.

Topic Maps. Topic Maps (TMs) [1] are an emerging new tool used for organisation, management, and navigation of large and interconnected corpora of information resources. TMs collect key concepts of information sources and tie them together. The key concepts in topic maps are topics, associations, and occurrences. A topic is a resource within the computer that stands in for (or 'reifies') some real-world subject. An association is a link used to tie related topics. Occurrences are a set of information resources that may be linked to a topic. Topics can have base name, and variants of base names. This naming scheme allows the applicability of different names to be used in different specific contexts or scopes such as for multilingual representations. Topics have types expressed by class-instance relation (Rome is instance of a city). Topic types are also topics according to TMs standard (ISO/IEC 13250-2000). The notion of scope (the limit of validity) in TMs is one of the key distinguishing features of the topic maps paradigm; scope makes it possible for topic maps to incorporate diverse worldviews. In a given context, a topic has a specific scope. Different TMs can form different layers above the same information pool and provides us with different views of it. In fact, TMs capture contextual validity through the aspects of scope. Another filtering mechanism used in TMs is the facet which is used to assign metadata to the information resources. Thus, facets are based on the properties of information resources whereas scope is based on the properties of the topics.

Object-Role. There has been a large amount of work in databases addressing specific facets of multi-representation. The first technique that one could think of is the well-known view mechanism. Views allow one to derive a new (virtual) representation from the representations already defined in the database. In relational DBMS the view mechanism is quite powerful (e.g., new, virtual relations may be built from any number of pre-existing relations). In object-oriented DBMS,however, views have less flexibility and are mostly restricted to simple selection views. Traditionally the generalisation link, which by definition links two representations (one more generic, one more specific) of the same real world entity, is supported in semantic and object-oriented approaches to describe several classifications of the same phenomenon. *Is-a links*, however, form static hierarchies that come with an associated inheritance mechanism, subtyping constraints and instantiation rules, that restrict modeling expressiveness. This restriction has prompted lots of works searching for a more flexible classification scheme. Those approaches assume that an analysis of the object life cycle determines the set of relevant possible representations for an object. These alternative representations, called roles, aspects or deputy objects, are described as types with specific rules, and more flexible inheritance mechanisms. These propositions allow, for instance, objects to move from one type to another or to acquire new instances as needed ([2,12,24,5,18]) or to be dynamically re-grouped to form new classes, while keeping their membership in their original classes (e.g., [16]). Another solution, ([20]), inspired by the work in temporal databases proposes to describe stamped multi-representation types. Stamps allows to characterize the different representations and also to filter access to data.

5 Conclusion

In this paper, we have presented a solution to the problem of multirepresentation ontologies. In this respect, we have illustrated how an ontological concept may be represented in several facets/contexts according to its intended uses. A contextual ontology language is presented. It is based on description logics and stamping technique. Specific constructors are adapted from the existing ones to deal with the multirepresentation aspect.

As future work, we aim at finalizing and implementing the proposed constructs. Further, we intend to validate and test the proposed language in the domain of urbanism where we expect a wide range of contexts like transportation, land use, urban planing, etc. A link with existing standard ontology languages based on description logic (OIL, DAML+OIL) will be established in the future phase of our study. Finally, we intend to adapt the class-based logic query language [6] to deal with the need of browsing and searching concepts in large multirepresentation ontologies.

References

1. ISO 13250. Information technology – sgml applications – topic maps(. Technical report, International Organization for Standardization, ISO/IEC 13250, 1995.
2. Albano A, Ghelli G, and Orsini R. Fibonacci. A programming language for object databases. *Very Large Data Bases Journal*, 4(3):403–444, 1995.
3. J. Angel, D. Fensel, and R.Studer. The model of expertise in karl. In *Proceedings of the 2nd World Congress on Expert Systems, Lisbon,/Estoril,Portugal*, January 1994.
4. Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, 9:463–506, July 1998. December 1998.
5. Pernici B. Objects with roles. In *In Proceedings of ACM Conference on Office Information Systems, Cambridge, Massachusetts*, pages 205–215, 90.
6. Djamal Benslimane, Mohand-Said Hacid, Evimaria Terzi, and Farouk Toumani. A class-based logic language for ontologies. In *In the proceedings of the Fifth International Conference on Flexible Query Answering Systems. October 27–29, 2002, Copenhagen, Denmark. Proceedings. Lecture Notes in Computer Science 1864 Springer 2000, ISBN 3-540-67839-5*, pages 56–70, October 2002.
7. M. Buchheit, F.M. Donini, and A. Scharaef. Decidable reasoning in terminological knowledge representation systems. *Journal of AI Research*, 1:109–138, 1993.

8. A. Delteil and catherine Faron-Zucker. Extending rdf(s) with contextual and definitional knowledge. In *In Proceedings of the IJCAI Workshop on E-Business and the intelligent Web, Seattle, 2001.*, 2001.
9. Fonseca F., Egenhofer M., Davis C., and Câmara G. Semantic granularity in ontology-driven geographic information systems. *Annals of Mathematics and Artificial Intelligence*, 1-2(36):121–151, July 2002.
10. A. Farquhar, R. Fikes, and J.P. Rice. The ontolingua server: A tool for collaborative ontology construction. In *Journal of Human-Computer Studies, 46:*, pages 707–728, 1997.
11. D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. Oil in a nutshell. In *In Proceedings of the ECAI-2000 Workshop on Applocations of Ontologies and Problem-Solving Methods, Berlin (Allemagne)*, 2000.
12. Gottlob G, Schrefl M, and Röck B. Extending object-oriented systems with roles. *In ACM Transactions on Information Systems*, 14(3):. 268–296, 1996.
13. M.R. Lee. Context-dependent information filtering. In *Proceedings of the International Symposium on Research, Development and Practice in Digital Libraries : ISDL'97 Conference*, November 1997.
14. O. LI and F.H. Lochovsky. Adome: An dvanced object modeling environment. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):255–276, March/April 1998.
15. Enrico Motta. An overview of the ocml modelling language. In *In Proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML'98), Karlsruhe, Germany*, pages 125–155, January 1998.
16. M.P. Papazoglou, B.J. Kramer, and A. Bouguettaya. On the representation of objects with polymorphic shape and behavior. In *In Proceedings. 13th International Conference on Entity-Relationship Approach, Manchester, UK*, pages 223–240, 94.
17. Christine Parent. Modeling spatial data in the mads conceptual model. In *Proceedings of the International Symposium on Spatial Data Handling,SDH98,Vancouver, Canada.*, July 1998.
18. J. Richardson and P. Schwartz. Aspects: extending objectsto support multiple, idependent roles. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 298–307, 1991.
19. Stefano Spaccapietra, Christine Parent, and Christelle Vangenot. Gis databases: From multiscale to multirepresentation. In *4th Int. Symp., SARA 2000, Texas, USA. Proc. LNCS 1864 Springer.*, pages 57–70, July 2000.
20. C. Vangenot, C. Parent, and S. Spaccapietra. Supporting decision-making with alternative data representations. volume 5, 2001. Available: http://www.geodec.org/.
21. W3C. Daml+oil reference description. Technical report, W3C, http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218, December 2001. W3C Note 18 December 2001.
22. W3C. Web ontology language (owl). Technical report, W3C, http://www.w3.org/TR/2002/WD-owl-ref-20021112/, November 2002. W3C Working Draft 12 November 2002.
23. F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In *In Proceedings of the 6th Int. Conf. on Knowledge Representation and Reasonning, KR'98, Trento, Italy, June 1998.*, pages 512–523, 2001.
24. Kambayashi Y and Peng Z. Object deputy model and its applications. In *. Proceedings of the Fourth International Conference on Database Systems for Advanced Applications, DASFAA'95, Singapore*, pages 1–15, 95.

Appendix: Syntactical and Semantics Aspects of (Non-contextual) Description Logics

Definition 3. (Syntax of concept terms) *Let $\mathcal{C}$ be a set of concept names, $\mathcal{R}$ be a set of role names and $n \in \mathcal{N}$. Non-contextual concept terms C and D can be formed by means of the following syntax:*

$$\begin{array}{rll}
C, D \longrightarrow & A \mid & \text{(atomic concept)} \\
& \top \mid \bot \mid & \text{(top, bottom)} \\
& C \sqcap D \mid & \text{(conjunction)} \\
& C \sqcup D \mid & \text{(disjunction)} \\
& \neg C \mid & \text{(complement)} \\
& \exists R.C \mid & \text{(existential quantification)} \\
& \forall R.C \mid & \text{(value restriction)} \\
& (\leq nR) \mid & \text{(at most number restriction)} \\
& (\geq nR) & \text{(at least number restriction)}
\end{array}$$

Definition 4. (Semantics) *The semantics of the language is given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ which consists of an interpretation domain $\Delta^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ maps each atomic concept $A \in \mathcal{C}$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $R \in \mathcal{R}$ to a subset $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concepts is inductively defined as follows:*

$$\begin{array}{l}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} \subseteq \top^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
\quad (\neg C)^{\mathcal{I}} = \top^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\ (\exists R.C)^{\mathcal{I}} = \{x \in \top^{\mathcal{I}} \mid \exists y : (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
\ (\forall R.C)^{\mathcal{I}} = \{x \in \top^{\mathcal{I}} \mid \forall y : (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \\
\ (\leq nR)^{\mathcal{I}} = \{x \in \top^{\mathcal{I}} \mid \quad \|\{y \mid (x, y) \in R^{\mathcal{I}}\}\| \leq n\} \\
\ (\geq nR)^{\mathcal{I}} = \{x \in \top^{\mathcal{I}} \mid \quad \|\{y \mid (x, y) \in R^{\mathcal{I}}\}\| \geq n\}
\end{array}$$

Let A be a concept name and let D be a concept term. Then $A = D$ is a terminological axiom (also called defined-as axiom). A terminology (*TBox*) is a finite set $\mathcal{T}$ of terminological axioms with the additional restriction that no concept name appears more than once in the left hand side of the definition.

An interpretation $\mathcal{I}$ is a model for a *TBox* $\mathcal{T}$ if and only if $\mathcal{I}$ satisfies all the assertions in $\mathcal{T}$.

Definition 5. (Subsumption) *Let C, D be two concepts. D subsumes C (for short $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. C is equivalent to D (for short $C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$, i.e., $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$.*

Extension of Compositional Information Systems Development for the Web Services Platform

Dmitry Briukhov, Leonid Kalinichenko, and Iliya Tyurin

Institute of Informatics Problems RAS
{brd,leonidk,turin}@synth.ipi.ac.ru

Abstract. The use of Web services on the World Wide Web is expanding rapidly to make applications interoperable in information systems (IS). Web services providing interfaces to information and software components are convenient entities for producing their compositions having Web service appearances. At the same time, most large scale enterprise solutions that are deployed today are composed of a combination of different technologies that go together to compose many diverse applications. An approach for compositional information systems development in a multi-technological framework including Web service components is discussed. This paper proposes to extend the SYNTHESIS method for compositional information systems development (CISD) to the world of Web services. The CISD method is intended for correct composition of existing components semantically interoperable in the context of a specific application. Originally, the CISD method has been developed for the object-oriented platforms (like CORBA, RMI, J2EE). In the CISD, an ontological model and canonical object model (the SYNTHESIS language) are used for the unified representation of the new application (specification of requirements) and of the pre-existing components. Discovery of components relevant to the application and producing their compositions is provided in frame of the domain ontology and the canonical object model. To apply the CISD method for Web services, the mapping of WSDL specifications into the canonical model is required. The basic steps of the approach for the information system compositional development applying Web services are demonstrated.

1 Introduction

As XML becomes a standard for data representation spread among various platforms, it becomes easier to make information and software components to interoperate in information systems (IS) applying Web services. Web Services Description Language (WSDL [19]), Universal Description, Discovery and Integration (UDDI [17]) and Simple Object Access Protocol (SOAP [15]) provide basic facilities for technical interoperability in the world of Web services.

Web services can be treated as components that can be re-used to build information systems. While technical interoperability is supported by the Web service architecture, semantic interoperability is to be provided by specific organization of compositional development. This paper[1] describes an approach to use the SYNTHESIS method for compositional information systems development (CISD) [2] over Web services. The CISD method is intended for correct composition of existing components semantically interoperable in the context of a specific application (defined by a specification of requirements). In contrast to technical interoperability (provided by middleware technologies, such as CORBA, or Java RMI) in CISD the interoperability is considered in a broader, semantic aspect.

Originally, the CISD method has been developed for the object-oriented middlewares (like CORBA, J2EE), but it can be extended also for other platforms. In this paper we present an extension of the compositional IS development method for the Web Services platform. To apply the CISD method for Web services we need to extend specifications of Web services with ontologies. Also, the mapping of WSDL specifications into the canonical model of the CISD (SYNTHESIS language) should be provided.

The paper is organized as follows. After brief characterization of the related work, a short introduction into the CISD method is given. The intention of this introduction is to make further examples in the text better readable. Section 4 presents the main principles of mapping the WSDL specification into the SYNTHESIS specification. In section 5 an extension of the CISD method for the Web services platform is demonstrated. The conclusion summarizes the results obtained.

2 Related Work

Web Services is an additional middleware solution that has emerged from efforts to perform distributed computing over the public Internet, particularly exchanges between enterprises. In spite of many of its attractive capabilities, Web Services is unsuitable for many important operations, including rapid multi-use services (e.g., database transactions), or as the internal mechanism linking parts of component-based applications. Therefore solutions are needed to enable applications in diverse technologies to interoperate using web services technology. A Web Services architecture should seamlessly integrate with other infrastructure facilities on an "integration broker" platform [21,20]. Components from different platforms are expected to be composable together with and into the web services.

The CISD [2] is a compositional multi-platform method that is in a good correspondence with the idea of the integration broker [21]. This paper shows how to extend this method to the Web Services. Web Services requires a conceptual modeling and architectural framework on top of existing Web Service standards and architectures. Current Web services standards enable publishing service descriptions and finding services on a match based criteria such as method signatures or service category. More promising directions include DAML-S [6] providing an ontology of services, the service profile for advertising and discovering services; the process model, which gives a detailed description of a service's operation.

Brief comparison of CISD with the known DAML-S based approaches follows. The METEOR-S work [18] states that current Web service discovery mechanism is

[1] The work was supported by the Russian Foundation for Basic Research (grants 01-07-90376 and 03-01-00821).

inefficient, as it does not support discovery based on the capability of the services and thus leading to a lot of irrelevant matches. This project [18] adds an ontology-based approach to organize registries, enabling semantic classification of all Web services based on domains. An approach for semantic publication and discovery using WSDL descriptions map inputs and outputs of Web services to ontological concepts. The operations themselves should be mapped to concepts, all inputs and outputs in the WSDL description should not only be mapped to concepts in the domain specific ontology but also grouped according to operations. Similar mapping of operations and their parameters to ontologies has been originally provided by CISD [2].

In [14] the semantic web initiative at W3C (such as DAML-S) and tools that may help bridge the gap between the current standard solutions and the requirements for advanced web service discovery are analyzed. The analysis concludes that DAML-S is still in its infancy and a lot of work has to be done in order to overcome its limitations and problems. The paper [12] reports results of DAML-S usage to describe capabilities of Web services so that they can find each other on the basis of the information that they provide. An approach for integrated Semantic Web technology for automating customized, dynamic discovery of Web services together with interoperation through semantic translation is presented in [11]. Alongside with promising results, the paper warns that achieving automated Web service composition requires a fundamental shift in industrial frameworks from executing predefined process models to computing and adapting execution plans from abstract objectives.

Main differences of CISD [2] and DAML-S oriented approaches [11,12,14] consist in the following. CISD is based on a refinement technique [1] and on an extensible typed model for definition of services and workflows. Basic assumption of the method is that specifications of user needs and specifications of existing services may differ significantly. Ontological conformance is only one step in discovery of services potentially relevant to the user requirements. Further steps include an attempt to reconcile the value, structural and typing discrepancies between specifications applying transformation functions [2,3]. Adapting of behaviors is possible to some extent due to the features of the refinement technique that tolerates significant differences between abstract specification of requirements and pre-existing concrete implementation of a service. Reasoning reported in [11] applies another technique - proof in a description logic.

Another difference is that the CISD method [2], instead of using a specialized model (like DAML-S), applies quite general modeling framework suitable for equivalent mapping into it of various existing modeling facilities. Various ontological and process specification facilities have been experienced [10,16]. An approach for a specific model (DAML-S) mapping into the general modeling framework of CISD has been presented in [10].

3 Fundamentals of the CISD Method

The main distinguishing feature of the CISD method is a creation of compositions of component specification fragments refining specifications of requirements. Refining specifications obtained during the compositional development, according to the refinement theory, can be used anywhere instead of the refined specifications of requirements without noticing such substitutions by the users.

The compositional development is a process of systematic manipulation and transformation of specifications. Type specifications of the SYNTHESIS language are chosen as the basic units for such manipulation. The manipulations required include decomposition of type specifications into consistent fragments, identification of reusable fragments (patterns of reuse), composition of identified fragments into specifications refining the requirements, justification of substitutability of the results of such transformations instead of the specifications of requirements. The compositional specification calculus [7], intentionally designed for such manipulations uses the following concepts and operations.

A *type reduct* R_T is a subspecification of an abstract data type T specification. The specification of R_T should be formed so that R_T becomes a supertype of T.

For identification of a fragment of existing component type that may be reused in implementation of another type, a common reduct for these types should be constructed. *Common reduct* for types *T1* and *T2* is such reduct R_{T1} of *T1* that there exists a reduct R_{T2} of T2 such that R_{T2} is a *refinement* of R_{T1}. R_{T2} is called a *conjugate* of the common reduct. Establishing a refinement ensures that R_{T2} can be correctly substituted everywhere instead of R_{T1}.

For creation of composition of identified reusable fragments into specification refining the specification of requirements the type calculus operations (such as meet, join and product) are used. The *meet* operation T_1 & T_2 produces a type T as an 'intersection' of specifications of the operand types. The *join* operation $T_1 \mid T_2$ produces a type T as a 'join' of specifications of the operand types [7].

Briefly, the CISD method consists in the following. To make analysis of an information system being developed, an existing object analysis and design methods [13] extended to achieve the completeness of specifications is applied. In particular, such method is extended with a means for ontological definitions of the application domains, for definition of type invariants and predicative specifications of operations. Then, at the design phase, using a repository of component specifications, a search of the components relevant to the specifications of requirements is undertaken. This search is based on the integrated ontological context of requirements and components. Then, identification of reusable fragments of components, including reconciliation of various conflicts between specifications, and composition of such fragments into specifications concretizing the requirements should be made. And finally, a property of refinement of requirements by such compositions should be justified.

4 Representation of WSDL Specifications in the SYNTHESIS Language

Mapping the Web Services specifications into the SYNTHESIS language [8] is required to extend the CISD method to Web services.

In the CISD, the SYNTHESIS language intends to provide for uniform (canonical) representation of heterogeneous data, programs and processes for their use as interoperable entities. Strongly typed, object-oriented subset of the language (that is required in this paper) contains a universal constructor of arbitrary abstract data types, a comprehensive collection of the built-in types, as well as type expressions based on the operations of type calculus.

All operations over typed data in the SYNTHESIS language are represented by functions. Functions are given by predicative specifications expressed by mixed pre- and post-conditions.

In the SYNTHESIS language the type specifications are syntactically represented by frames, their attributes - by slots of the frames. Additional information related to attributes can be included into metaslots. Syntactically frames are embraced by figure brackets { and }, slots are represented as pairs <*slot name*> *:* <*slot value*> (a frame can be used as a slot value), slots in a frame are separated by semi-colons.

4.1 Port Type

A port type describes a set of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a set of input and output messages, a port type is a set of operations.

A port type can optionally extend one or more other port types. In such cases the port type contains the operations of the port types it extends, along with any operations it defines.

The specification of port type is the following:

```
<wsdl:portType name="port_name">
    <wsdl:documentation .... /> ?
    <wsdl:operation name="operation_name"> *
    ...
    </wsdl:operation>
</wsdl:portType>
```

The port type is mapped into the SYNTHESYS type:

```
{port_name;
  in: type;
  <list_of_operations>;
}
```

4.2 Port Type Operation

A port type operation describes an operation that a given port type supports. An operation is a set of message references. Message references may be to messages this operation accepts, that is input messages, or messages this operation sends, that is output or fault messages.

```
<wsdl:operation name="operation_name">
    <wsdl:documentation .... /> ?
    <wsdl:input message="input_message_name"> ?
        <wsdl:documentation .... /> ?
    </wsdl:input>
    <wsdl:output message="output_message_name"> ?
        <wsdl:documentation .... /> ?
    </wsdl:output>
    <wsdl:fault name="ncname"
                message="fault_message_name"> *
        <wsdl:documentation .... /> ?
    </wsdl:fault>
</wsdl:operation>
```

The port type operation is mapped into the SYNTHESIS function:

```
operation_name: {
  in: function;
  params: {<list_of_parameters>};
  [{{<predicative_specification>}}]
};
```

4.3 Message

A message describes the abstract format of a particular message that a Web service sends or receives. The format of a message is typically described in terms of XML element and attribute information items.

A part constituent describes a portion of a particular message that a web service sends or receives. The format of a part is described by reference to type definition or element declaration constituents.

```
<wsdl:message name="message_name">
    <wsdl:documentation .... /> ?
    <part name="part_name" element="element_name"?
          type="type_name"?/> *
</wsdl:message>
```

The message is mapped into a list of SYNTHESIS function parameters. Each part constituent is mapped into separate parameter:

```
operation_name: {in: function;
  params: {<kind>part_name/type_name, ...};
};
```

or

```
operation_name: {in: function;
  params: {<kind>part_name/element_name, ...};
};
```

The <kind> of parameter ("+" - input, "-" - output, "" - input-output) corresponds to message exchange pattern in WSDL.

4.4 Service

A service describes the set of port types that a service provides and the ports they are provided over.

```
<wsdl:serviceType name="service_name">
    <wsdl:portType name="port_name"/> +
</wsdl:serviceType>
```

The service is mapped into SYNTHESIS module which contains types corresponding to WSDL port types:

```
{service_name;
  in: module;
  type: {
    {port_name;
      in: type;
      ...
    }
  }
}
```

4.5 XML Schema Types

At the abstract level, the Types Component is a container for imported and embedded schema components. By design, WSDL supports any schema language for which the syntax and semantics of import or embed have been defined. Support for the W3C XML Schema Description Language is required of all processors. Principles of mapping the XML Schema Datatypes into the SYNTHESIS types are described in [4].

4.6 Example

To demonstrate the mapping we consider the *HotelReservationService*. Its specification in WSDL looks as follows:

```
<wsdl:types>
  <xsd:simpleType name="SetOfHotels">
    <list itemType="HotelAndPrice" />
  </xsd:simpleType>
  <xsd:complexType name="HotelAndPrice">
    <xsd:element name="hotel" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="roomType" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
  </xsd:complexType>
</wsdl:types>

<message name="InputAsk">
  <part name="city" type="xsd:string"/>
  <part name="roomType" type="xsd:string"/>
  <part name="fromDate" type="xsd:date"/>
  <part name="toDate" type="xsd:date"/>
</message>
<message name="OutputAsk">
```

```
  <part name="return" type="SetOfHotels"/>
</message>
<message name="InputComments">
  <part name="comment" type="xsd:string"/>
</message>

<portType name="HotelReservation">
  <operation name="askForRoom">
    <input message="tns:InputAsk"/>
    <output message="tns:OutputAsk"/>
  </operation>
  <operation name="sendComments">
    <input message="tns:InputComments"/>
  </operation>
</portType>

<serviceType name="HotelReservationService">
  <portType name="HotelReservation" />
</serviceType>
```

The specification of this Web service in SYNTHESIS is:

```
{HotelReservationService;
  in: module;
  type: {
    {HotelReservation;
      in: type;
      askForRoom: {in:function;
        params: {+city/string, +roomType/string,
          +fromDate/time, +toDate/time,
          -return/SetOfHotels;};
      };
      sendComments: {in:function;
        params: {+comment/string};
      };
    };
    {HotelAndPrice;
      in: type;
      hotel: string;
      city: string;
      roomType: string;
      price: integer;
    };
    {SetOfHotels: {set; type_of_element:
                               HotelAndPrice;};
    };
  }
}
```

5 Compositional Development over the Web Services Platform

The approach for compositional IS development over the Web services platform consists in an extension of the CISD [2]. Compositional design according to such extended approach includes the following steps:

- mapping the Web services specifications into the SYNTHESIS language;
- searching for relevant Web services;
- identifying fragments of existing Web services, that may be reused in IS being developed;
- resolve conflicts between specifications;
- composition of such fragments into specification refining the specification of IS;
- implementation of IS.

To illustrate this approach the following example will be used. Consider the development of IS type *TravelService* over two existing components (implemented as Web services): *AirTicketReservationService* and *HotelReservationService.*

TravelService as a specification of requirements contains operations: *getHotelInfo*, *getAirTicketInfo* and *sendComments*. The specification of *TravelService* type in SYNTHESIS looks as follows:

```
{TravelService;
  in: type;
  getHotelInfo: {in:function;
    params: {+info/Travel,
             -returns/{set; type_of_element: Hotel;}
    };
  };
  getAirTicketInfo: {in:function;
    params: {+info/Travel,
             -returns/{set; type_of_element: Ticket;}
    };
  sendComments: {in:function;
    params: {+comment/string}
  };
};
{Travel;
  in: type;
  fromCity: string;
  toCity: string;
  fromDate: time;
  toDate: time;
  roomType: string;
  flightClass: string;
  person: Person;
};
```

AirTicketReservationService contains the *AirTicketReservation* port type, which contains *getAvailableTickets* and *sendComments* operations. *HotelReservationService* contains the *HotelReservation* port type, which contains *askForRoom* and *sendComments* operations. The mapping of WSDL specification of *HotelReservationService* Web service into the SYNTHESIS language is given in section 4.6.

5.1 Searching for Relevant Web Services

Specifications of requirements and pre-existing components must be associated with ontological contexts defining concepts of the respective subject areas. Ontological concepts are described with their verbal definitions similar to definitions of words in an explanatory dictionary. Fuzzy relationships between concepts of different contexts are established by calculating correlation coefficients between concepts on the basis of their verbal definitions. The correlation coefficients are calculated using the vector-space approach [2].

For each constituent (service type, port type, message) defined in WSDL, the corresponding ontological concept is defined. Its verbal definition is taken from the *documentation* element that should be defined inside each WSDL constituent. Due to the correlation established, ontologically relevant service components can be found.

5.2 Identifying Relevant Fragments

This step consists in identifying among probably relevant types (corresponding to existing Web services) those that may be used for the concretization of the types of specification of requirements.

For identification of a fragment of existing component type that may be reused in implementation of another type, the common reduct for these types should be constructed. The specification of common reduct for types *TravelService* and *HotelReservation* is the following:

```
{R_TS_HR;
  in: reduct;
    metaslot
      of: TravelService;
      taking: {getHotelInfo, sendComments};
      c_reduct: CR_TS_HR;
    end;
};
```

A slot *of* refers to the reduced type *TravelService*. A list of attributes of the reduced type in the slot *taking* contains names of its attributes and functions that are to be included into the reduct. In our case the reduct includes functions *getHotelInfo* and *sendComments*.

Its conjugate is specified as a *concretizing reduct* that incorporates the correspondence between attributes and functions of a reduct being refined and its refinement, as well as the required functions reconciling conflicts between specifications of these reducts:

```
{CR_TS_HR;
  in: c_reduct;
    metaslot
      of: HotelReservation;
      taking: {askForRoom};
      reduct: CR_TS_HR;
    end;
  simulating: {
    R_TS_HR.getHotelInfo(info) ~
        CR_TS_HR.askForRoom(info.toCity, info.roomType,
                            info.fromDate,info.toDate),
    R_TS_HR.sendComments(comment) ~
        CR_TS_HR.sendComments(comment)
  };
};
```

Slot *reduct* refers to the respective common reduct. A slot *of* refers to the reduced type *HotelReservation*. A list of attributes of the reduced type in the slot *taking* contains names of its attributes and functions that are to be included into the concretizing reduct. In our case it includes functions *askForRoom* and *sendComments*. Predicate *simulating* shows how the common reduct state is interpreted by an instance of the collection. E.g., it defines that function *getHotelInfo* of reduct *R_TS_HR* is refined by function *askForRoom* of concretizing reduct *CR_TS_HR* and shows the correspondence of their parameters. The concretizing reduct may also contain functions resolving various conflicts between specifications of the considered types.

In the same way the common reduct *R_TS_ATR* and concretizing reduct *CR_TS_ATR* for types *TravelService* and *AirTicketReservation* are specified. Reduct *R_TS_ATR* contains operations: *getAirTicketInfo* and *sendComments*.

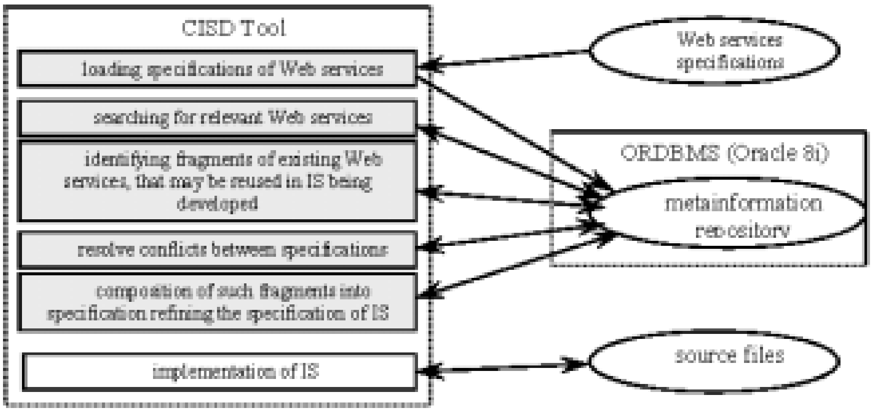

CISD Tool
loading specifications of Web services
searching for relevant Web services
identifying fragments of existing Web services, that may be reused in IS being developed
resolve conflicts between specifications
composition of such fragments into specification refining the specification of IS
implementation of IS
Web services specifications
ORDBMS (Oracle 8i)
metainformation repository
source files

Fig. 1. CISD Tool Structure

5.3 Composition of Relevant Fragments

For creation of composition of identified fragments into specification refining the specification of requirements the type calculus operations (such as meet, join and product) [7] are used.

In our example for implementing the *TravelService* type the join of reducts of *AirTicketReservation* and *HotelReservation* types is used. The specification of concretizing type *CT_TS_HR_ATR* is constructed as join of reducts *R_TS_HR* and *R_TS_ATR*:

```
CT_TS_HR_ATR = R_TS_HR[getHotelInfo, sendComments] |
               R_TS_ATR[getAirTicketInfo, sendComments]
```

Resulting type *CT_TS_HR_ATR* contains functions: *getHotelInfo*, *getAirTicketInfo* and *sendComments*. The concretizing type refines the *TravelService* type.

5.4 Implementation of the Developing IS

During implementation the results obtained at previous steps are transformed into the form of source files in the programming language. Implementation of the IS being developed includes the code generation for constructed reducts and concretizing types. Each reduct and concretizing type is implemented as a separate class. Code generation is performed using special macro language. Programs written in the macro language may be customized to meet particular needs of specific implementation platforms.

5.5 CISD Tool

Figure 1 shows a general structure of the tool supporting compositional IS development. The tool is being developed reusing the SYNTHESIS compositional IS design method prototype [2]. The tool is based on Oracle 8i and Java 2 under Windows environment.

In figure 2 an example of CISD Tool GUI is presented. It shows the GUI for confirmation of relevance between the specification of requirements element (function *getHotelInfo*) and the component element (function *askForRoom*).

6 Conclusion

The paper presents an approach for information systems development as a process of composition of existing Web services. It is based on the compositional information systems development method described in [2]. To extend this method to Web services, the mapping of WSDL specifications into SYNTHESIS specifications was introduced. The basic steps of constructing Web services composition refining a specification of requirements were demonstrated. The approach proposed makes possible composition of components developed in frame of different technologies (Web

Services, CORBA, J2EE) that work together to compose many diverse applications. Specific integration architecture [20,21] is assumed for the implementation.

In the future we plan to extend presented approach with process specifications of Web services (in spirit of process reuse [9] represented similarly to WS-BPEL [5] or DAML-S [6], but based on the refinement and bisimulation technique).

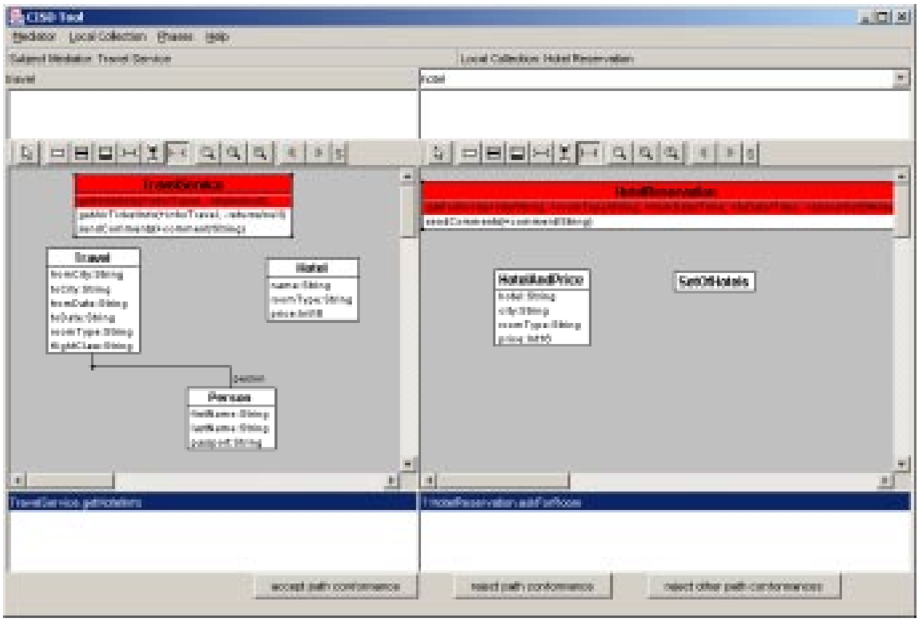


Fig. 2. An Example of CISD Tool GUI

References

[1] Abrial J.-R. The B Book: assigning programs to meaning. Cambridge University Press, 1996

[2] Briukhov D.O., Kalinichenko L.A. Component-Based Information Systems Development Tool Supporting the SYNTHESIS Design Method. In *Proc. of the East European Symposium on "Advances in Databases and Information Systems"*, Poland, Springer, LNCS No.1475, 1998

[3] Briukhov D.O., Kalinichenko L.A., Skvortsov N.A., Stupnikov S.A. Value Reconciliation. Mediators of Heterogeneous Information Collections Applying Well-Structured Context Specifications In Proceedings of the Fifth International Baltic Conference on Databases and Information Systems BalticDB&IS'2002, Tallinn, Estonia, June 3-6, 2002

[4] Briukhov D.O., Tyurin I.N. Mapping XML Schema Data Types into SYNTHESIS Types (in Russian). In *Proc. of the Russian Conference "Digital Libraries: Advanced Methods And Technologies, Digital Collections" (RCDL'2002)*, Dubna, 2002

[5] Business Process Execution Language for Web Services, Version 1.0. http://www-106.ibm.com/developerworks/library/ws-bpel/

[6] *DAML-S 0.7 Draft Release*. http://www.daml.org/services/daml-s/0.7/

[7] Kalinichenko L. A. Compositional Specification Calculus for Information Systems Development. In *Proc. of the East-West Symposium on Advances in Databases and Information Systems (ADBIS'99)*, Maribor, Slovenia, September 1999, Springer Verlag, LNCS, 1999

[8] Kalinichenko L. A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of informatics, Russian Academy of Sciences, Moscow, 1995

[9] Kalinichenko L.A. *Workflow Reuse and Semantic Interoperation Issues. Advances in workflow management systems and interoperability.* A.Dogac, L.Kalinichenko, M.T. Ozsu, A.Sheth (Eds.),. NATO Advanced Study Institute, Istanbul, August 1997

[10] L.A. Kalinichenko, N.A. Skvortsov Extensible Ontological Modeling Framework for Subject Mediation . Proceedings of the Fourth All-Russian Conference on Digital Libraries, RCDL'2002, Dubna, October 15–17, 2002

[11] Daniel J. Mandell, Sheila A. McIlraith. A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[12] Massimo Paolucci, Katia Sycara, Takuya Nishimura, Naveen Srinivasan. Toward Semantic Web Services. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[13] *Paradigm Plus Reference Manual*. Protosoft, 1997

[14] Thomi Pilioura, Aphrodite Tsalgatidou , Alexandros Batsakis. Using WSDL/UDDI and DAML-S in Web Service Discovery. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[15] *Simple Object Access Protocol (SOAP) 1.1.* W3C Note 08 May 2000. http://www.w3.org/TR/SOAP/

[16] Stupnikov S.A., Kalinichenko L.A., Jin Song DONG Applying CSP-like Workflow Process Specifications for their Refinement in AMN by Pre-existing Workflows In Proceedings of the Sixth East-European Conference on Advances in Databases and Information Systems ADBIS'2002, September 8-11, 2002, Bratislava, Slovakia

[17] *UDDI Version 3.0 Specification.* http://uddi.org/pubs/uddi_v3.htm

[18] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. and Miller, J. METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (to be published)

[19] *Web Services Description Language (WSDL) Version 1.2.* http://www.w3.org/TR/wsdl12

[20] *Web Services Software Architecture (WSSA)*, RFI, OMG Document # bei/2003-01-04, January 6, 2003

[21] *White Paper on Web Services Integration Architecture*, October 28, 2002, OMG Document Numbers:bei/2002-10-02

Domain Based Identification and Modelling of Business Component Applications

Antonia Albani, Alexander Keiblinger, Klaus Turowski, and Christian Winnewisser

Chair of Business Information Systems (Wirtschaftsinformatik II)
University of Augsburg
Universitätsstraße 16, 86135 Augsburg, Germany
{antonia.albani, alexander.keiblinger, klaus.turowski, christian.winnewisser}@wiwi.uni-augsburg.de

Abstract. This paper presents a process for the design of domain specific business component applications to enforce the use of component based software technologies. Starting from the notion that the direct mapping of business processes to business components often leads to inadequate results, a methodology for identifying and refining business components based on the functional decomposition of an application domain will be presented. The usability of the resulting process will be shown with the example of a concept for strategic supply chain development, which extends the traditional frame of reference in strategic sourcing from a supplier-centric to a supply-chain-scope including the dynamic modelling of strategic supply-chains.

1 Introduction

The idea of software systems made up from pre-fabricated software components that could be exchanged via software component markets has been on the agenda of software engineering at least since McIlroy has outlined his vision in 1968 [15]. The underlying idea is to combine components from different vendors to an application which is individual to each customer and where the compositional plug-and-play-like reuse of black box components enables software component markets. Ideally, the advantages of both standard and individual software production are combined. Compositional reuse is a special kind of reuse technique as *generative* techniques [7] or *code and design scavenging* [19, pp. 25–28]. The principle of modular design that is underlying component based software systems is equally important for the discussion of the technological as well as the economic advantages of component based software systems. A rich literature on the general advantage of such systems exists, c.f. [4], [3], [20], [21], [23]. Modular systems have been described as the result of a functional decomposition [32] and the conception of modular systems has been thoroughly analysed by system theory.

According to [8, pp. 3–4], we define the term *component* as follows: A component consists of different (software-) artefacts. It is reusable, self-contained and marketable, provides services through well-defined interfaces, hides its implementation and can be deployed in configurations unknown at the time of development. A *business* *component* is a component that implements a certain set of services out of a given business domain. Refer to [27 pp. 164–168] and [8] for an in depth discussion of various other component approaches given in literature.

For the concept of business components as described above, a standard approach for specification has been defined in the memorandum “Standardized Specification of Business Components” [31]. In section 2 the specification will be briefly explained. Whereas an established approach for the specification of business components already exists, no methodology for the identification of reusable, marketable and self-contained business components has been defined so far. Therefore, in section 3 a business component modelling process is proposed for the identification of suitable business components. Component based domain analysis and component based domain design with the resulting models are introduced and described in detail.

To illustrate and evaluate the proposed methodology, the business domain of strategic supply chain development is introduced. Based on this domain scope a business component model has been identified by applying the business component modelling process. The phases of the respective component based domain analysis will be described in section 4. In section 5 the component based domain design will be illustrated.

2 Business Components

For the use of business components it is necessary to *specify* them, i.e., standardise them in a domain dependent context and describe their interface and behaviour in a consistent and unequivocal way. Such a specification is crucial for the creation of component markets, since component buyers are unable to specify their demands or to cross check offered components against their demands without it [10].

The memorandum “Standardized Specification of Business Components” [31] is a recommendation of the working group *business components* at the German Informatics Society (GI). It aims to set a *methodical standard* for the specification of business components enabling a common understanding of component specifications. This is achieved by identifying the objects to be specified and by defining a notation mix that is standardised, accepted and agreed by all participating parties. This ensures the reusability of components and their exchange between companies and software developers can be simplified. The s*pecification* of a business component is defined as a complete, unequivocal and precise description of its external view. It describes which services a business component provides under which conditions.

Based on the ideas of [5], [30] and [29], the memorandum states that the specification of business components is to be carried out on different contract levels (see Fig. 1). Besides arranging the specifications’ contents according to contract levels, a specific notation language is proposed for each level of abstraction.

In the context of systematic specification of business components it is helpful to use a well-known and well-accepted formal notation, which can be used on more than one contract level. A notation is called *formal* in this context if syntax and semantics of the notation are unequivocal and consistent. For this reason, formal notations seem to be particularly suitable for the specification of software contracts. A detailed characterisation of the different contract levels is given in [31].

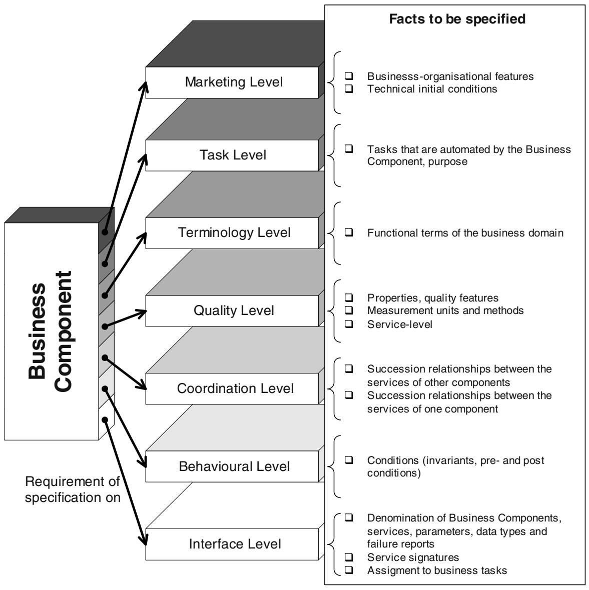

Business Component
Requirement of specification on
Marketing Level
Task Level
Terminology Level
Quality Level
Coordination Level
Behavioural Level
Interface Level
Facts to be specified
Businesss-organisational features
Technical initial conditions
Tasks that are automated by the Business Component, purpose
Functional terms of the business domain
Properties, quality features
Measurement units and methods
Service-level
Succession relationships between the services of other components
Succession relationships between the services of one component
Conditions (invariants, pre- and post conditions)
Denomination of Business Components, services, parameters, data types and failure reports
Service signatures
Assigment to business tasks

Fig. 1. Software contract levels and facts to be specified

3 Business Component Modelling Process

A precondition to component based development of application systems by using business components is a stable component model. In order to obtain stable business component models, a well defined derivation process for such components is necessary. Based on the fact that business components do not only satisfy the requirements for a single application system but rather for a family of systems – and therefore for a certain domain – the derivation process requires throughout all development phases the consideration of the specific business domain.

Domain Engineering [7 , p. 19–59, 19 , p. 159–169] aims at the development of reusable software and a large number of Domain Engineering processes exist, among the most important ones being [11, 26]. The methods mentioned contribute to different aspects of the Domain Engineering area as for example in identifying prominent or distinctive features of a class of systems or in defining maintainability and understandability characteristics of family of systems.

For business components additional prerequisites are required, the most important ones being reusability, marketability and self-containment for different domains. Therefore a new methodology for business components – the Business Component Modelling (BCM) process – is introduced next. In section 4 and 5 the BCM is used to define suitable business components for the domain of Strategic Supply Chain development (SSCD).

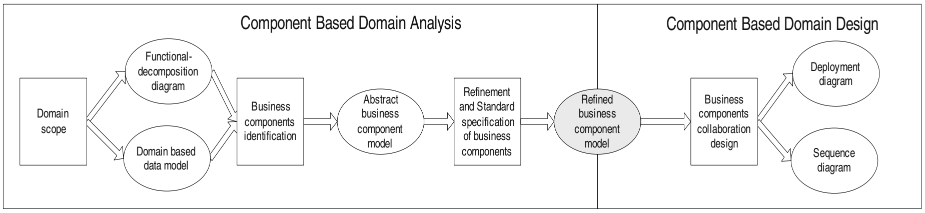

Component Based Domain Analysis
Component Based Domain Design
Domain scope
Functional-decomposition diagram
Domain based data model
Business components identification
Abstract business component model
Refinement and Standard specification of business components
Refined business component model
Business components collaboration design
Deployment diagram
Sequence diagram

Fig. 2. Business Component Modelling process

An overview of the BCM process focusing on the *Component Based Domain Analysis* and *Component Based Domain Design* phases is given in Fig. 2. Rectangles denote sub phases of the process and ellipses contain the resulting diagrams and models of each sub phase.

In the following the single phases and their performed tasks are explained. During *domain scope* the domain of interest is identified, characterised and business processes with their functional tasks are defined. In addition data is collected to analyse the information objects and their relationships. Possible sources of domain information include existing systems in the domain, domain experts, handbooks, requirements on future systems, market studies, and so on. As a result of the first sub phase a *functional-decomposition diagram* and a *domain based data model* are generated. This information is fundamental for the next sub phase in the BCM process, namely the *Business Components Identification.*

Having defined the functional-decomposition diagram the most intuitive way of identifying business components would be the mapping of single business processes to business components and their functional tasks to business component services. Having experienced that, the resulting components are not satisfactory. They are often not reusable, self-contained or marketable as postulated in [31], and the quality of services does not satisfy criteria of performance engineering – e.g. adaptable, performant, maintainable – as described in [25, 24]. In order to optimise the process of identifying high quality, reusable and marketable business components the Business Components Identification (BCI) method is introduced.

BCI is based upon the Business System Planning (BSP) [1] method and has been modified for the field of business components identification. BCI takes as input the business tasks of a specific domain, as defined in the functional-decomposition diagram, and the domain based data model, both obtained from the domain scope phase. In a first step a matrix is built defining the relationships between the single business tasks and the informational data. The relationships are visualised inserting “C” and “U” in the matrix. “C” denotes that the data is *created* by the specific business task, and “U” denotes the *usage* of informational data by a given task. In changing the order of data and of business tasks and placing the “C” as far left and as up as possible in the matrix, groups of relationships can be recognised, see Table 2 and Table 3. These groups identify potential business components. If some “U”’s are outside of the groups, arrows are used to identify the data flow from one group to the other. The result of the BCI is an abstract business component model with some already defined dependencies between components. In section 4 the BCI method is explained by means of the domain of strategic supply chain development.

In the next sub phase each component of the abstract model needs to be refined and specified as stated in the standardised specification of business components document. An overview of the specification levels has been given in section 2. The result is a *refined business component model* with all seven business component levels – marketing, task, terminology, quality, coordination, behavioural and interface – specified.

The refined business component model is the main input for the *Component Based Domain Design* and *Component Based Domain Implementation* phases. The component based domain design phase is explained next, whereas the implementation phase is not within the scope of this paper to be defined.

The sub phase *business components collaboration design* of the design phase is used to define the deployment of component instances on different systems and to identify dependencies in form of service calls between business components. As a result different kinds of diagrams are produced, e.g. deployment and sequence diagrams. These diagrams together with the refined business component model are the outcome of the BCM process and are the main input for developing component based information systems with high quality, reusable and marketable business components. The complete BCM process is summarised in Table 1.

Table 1. Summary of the BCM Process

BCM Phase	BCM sub phases	Performed Tasks	Results
Component Based Domain Analysis	Domain scope	Identification and characterisation of the domain	Functional-decomposition diagram
		Definition of the business processes and functional tasks of the domain	
		Data collection and definition of information objects	Domain based data model
		Identification of relationships between information objects	
	Business Component Identification (BCI)	Grouping of functional business tasks and informational object for the identification of business components	Abstract business component model
	Standard specification of business components	Specification of all business component levels (marketing, task, terminology, quality, coordination, behaviour, interface)	Refined business component model
Component Based Domain Design	Business components collaboration design	Definition of component instances	Deployment diagram
		Definition of dependencies between component instances	
		Identification of service calls between component instances	Sequence diagram

To illustrate the process of BCM it is applied in the following sections to the domain of strategic supply chain development detailing both phases, sub phases, tasks and the resulting diagrams and models.

4 Component Based Domain Analysis

The goal of the component based domain analysis is to generate a refined business component model with all business components being specified as defined in the memorandum for standardised specifications of business components.

4.1 Domain Scope – From Strategic Sourcing to Strategic Supply Chain Development

The relevance of the purchasing function in the enterprise has increased steadily over the past two decades. Till the 70ies, purchasing was widely considered an operative task with no apparent influence on long term planning and strategy development [16]. This narrow view was broadened by research that documented the positive influence that a targeted supplier collaboration and qualification could bring to a company's strategic options [2]. In the 80ies, trends such as the growing globalisation, the focus on core competencies in the value chain with connected in-sourcing and out-sourcing decisions, as well as new concepts in manufacturing spurred the recognition of the eminent importance of the development and management of supplier relationships for gaining competitive advantages. As a result, purchasing gradually gained a strategic relevance on top of its operative tasks [12].

Based on these developments, purchasing has become a core function in the 90ies. Current empiric research shows a significant correlation between the establishment of a strategic purchasing function and the financial success of an enterprise, independent from the industry surveyed [6, p. 513]. One of the most important factors in this connection is the buyer-supplier-relationship. At many of the surveyed companies, a close cooperation between buyer and supplier in areas such as long-term planning, product development and coordination of production processes led to process improvements and resulting cost reductions that were shared between buyer and suppliers [6, p. 516].

In practice, supplier development is widely limited to suppliers in tier-1. With respect to the above demonstrated, superior importance of supplier development we postulate the extension of the traditional frame of reference in strategic sourcing from a supplier-centric to a supply-chain-scope, i.e., the further development of the strategic supplier development to a strategic supply chain development. This re-focuses the object of reference in the field of strategic sourcing by analysing supplier networks instead of single suppliers. Embedded in this paradigm shift is the concept of the value network that has been comprehensively described, e.g., [14], [33], [18].

The main reason for the lack of practical implementation of strategic supply chain development can be found in the high degree of complexity that is connected with the identification of supply chain entities and the modelling of the supply chain structure, as well as the high coordination effort, as described by [13].

In a next step the functional tasks of strategic supply chain development have to be defined. Those tasks will be derived from the main tasks of strategic sourcing. The most evident changes are expected for functions with cross-company focus. The functional tasks of strategic supply chain development have been illustrated in a function decomposition diagram (Fig. 3).

Processes and tasks that will be included in the component model have been shaded. Following, selected tasks will be described. The focus will be on changes to current tasks of strategic purchasing.

Task "Model strategic supply chain": The process *supplier selection* from strategic purchasing undergoes the most evident changes in the shift to a supply chain centric perspective. The expansion of the traditional frame of reference in strategic sourcing requires more information than merely data on existing and potential suppliers in tier-1. Instead, the supply chains connected with those suppliers have to be identified and evaluated, e.g., by comparing alternative supply chains in the production network. As a consequence, the task *supplier selection* is part of the process that leads to the modelling of strategic supply chains.

The perception of the supply chain as dynamic network constitutes the basis for the identification of strategic supply chains. To visualise the network in a structured way, a specific strategic demand is communicated to existing and/or potential suppliers in tier-1. Subsequently, those tier-1 suppliers report their own demand to their respective suppliers. The requested information are split-lot transferred the other way round, aggregated and finally visualised as supply network, in which each participant of the supply chain constitutes a network hub. Without information technology, such an approach could not be realised.

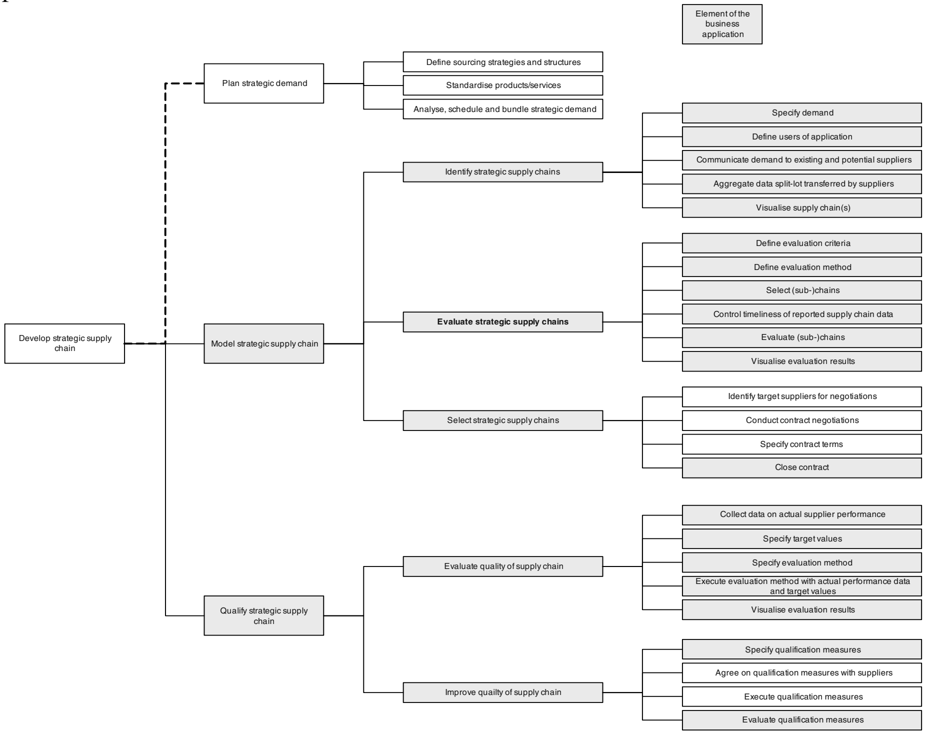

Element of the business application
Develop strategic supply chain
Plan strategic demand
Define sourcing strategies and structures
Standardise products/services
Analyse, schedule and bundle strategic demand
Model strategic supply chain
Identify strategic supply chains
Specify demand
Define users of application
Communicate demand to existing and potential suppliers
Aggregate data split-lot transferred by suppliers
Visualise supply chain(s)
Evaluate strategic supply chains
Define evaluation criteria
Define evaluation method
Select (sub-)chains
Control timeliness of reported supply chain data
Evaluate (sub-)chains
Visualise evaluation results
Select strategic supply chains
Identify target suppliers for negotiations
Conduct contract negotiations
Specify contract terms
Close contract
Qualify strategic supply chain
Evaluate quality of supply chain
Collect data on actual supplier performance
Specify target values
Specify evaluation method
Execute evaluation method with actual performance data and target values
Visualise evaluation results
Improve quality of supply chain
Specify qualification measures
Agree on qualification measures with suppliers
Execute qualification measures
Evaluate qualification measures

Fig. 3. Functional decomposition diagram for the supply chain development

According to the assumptions described above, the rating of supply chains requires the evaluation of networks instead of single suppliers. There has been preparatory work on evaluation methods for business networks (e.g., [28, 22]) on which we have based initial methods for the described application. However, there is need for further research, especially in the area of aggregation of incomplete information.

Task "Qualify strategic supply chain": In addition to the selection of suitable supply chains and composition of alternative supply chains, the performance improvement of strategically important supply chains is one of the major goals of strategic supply chain development. Main prerequisite is the constant evaluation of the actual performance of selected supply chains by defined benchmarks. The application should support respective evaluation methods and enables the user to identify imminent problems in the supply chain and to initiate appropriate measures for qualification of supply chain partners.

This is important because of the long-term character of strategic supply chain relationships. As a result of the long-term perspective, qualification measures – e.g., along the dimensions product, processes and management abilities – require deployment of resources on the buyer side as well. Because of this effort, problems in the supply chain should be identified proactively and qualification measures should be tracked.

Task "Plan strategic demand": Strategic planning, i.e., analysing, scheduling and grouping of long-term demand, primarily affects intra-company processes that will not change significantly by switching to a supply chain perspective in strategic purchasing and therefore will not be automated in a first step.

Data Model

Having identified and characterised the domain of strategic supply chain development and defined the business processes and tasks of that domain, it is necessary to determine the relevant information objects involved in the modelling of the strategic supply chain development. The resulting data model is listed as UML class diagram [17 p. 294] in Fig. 4.

Starting from the demand of a *buyer* it is relevant to collect data not only from the suppliers in tier-1, but from all suppliers in tier-n. A *demand* consists of *services*, *material groups* and more which can be generalised to a *demand category*. To each demand category different *characteristics* can be added, as for example *method of production*, *service level*, *time*, *volume* etc. A whole supply chain specified by a demand is a network of suppliers providing information to the buyer which is used for the development of the supply chain. This network of suppliers is represented in the data model as a *complex monitoring object* whereas a single supplier is represented as an *elementary monitoring object*. With the affiliation of elementary monitoring objects to a complex monitoring object and with the identification of predecessors and successors of such elementary monitoring objects the whole supply chain is defined. At a particular time each elementary object provides information about the *product range, bill of material, financial data* or more. This information is known as *supplier generated data*. In addition the buyer generates own data, called *buyer generated data*, specifying the performance of the supplier, respectively of the elementary monitoring object, as termed in the data model. Examples for data generated by the buyer are *target performance data* and *actual performance data*. Target performance data are guidelines for the supplier, and the actual performed data are the work performed measured by the buyer. The buyer holds with the acquisition of supplier generated data and with the definition and the measurements of performance data for all different complex monitoring objects the information needed to evaluate the supply chain. Different evaluation methods are defined by different evaluation criteria.

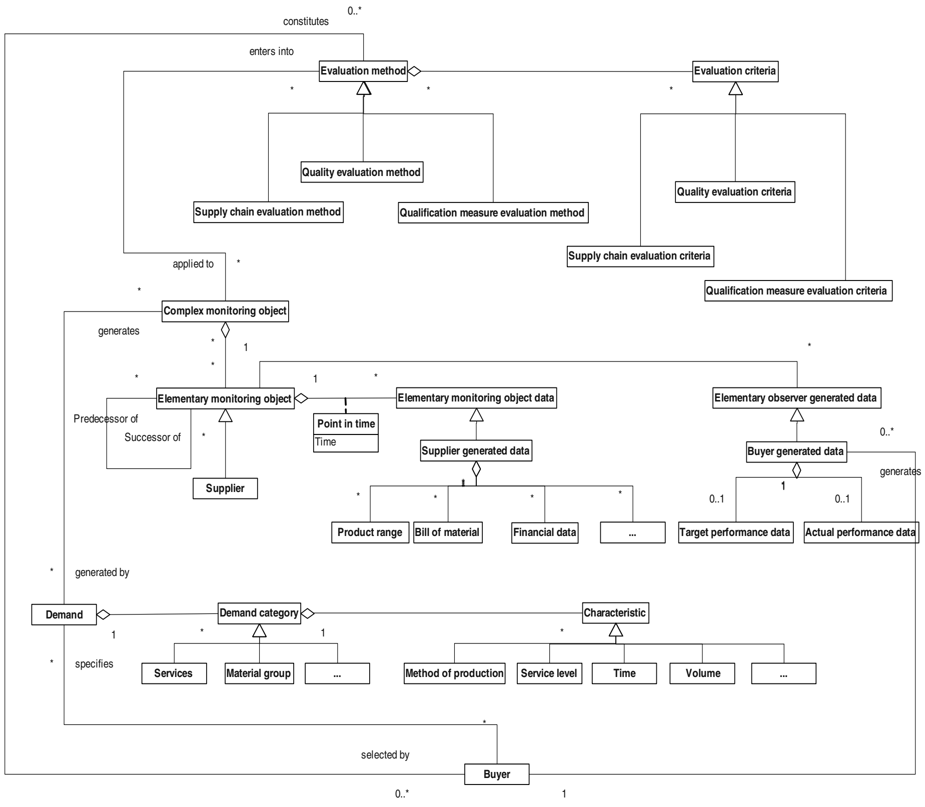


Fig. 4. Data model for the domain of strategic supply chain development

The data model and the functional-decomposition diagram derived from the domain scope are the main information needed for the identification of high quality business components explained in the next subsection.

4.2 Business Components Identification (BCI)

With the BCI method relationships between business tasks and information objects are defined and grouped. In Table 2 the relationships for the domain of strategic supply chain development are shown.

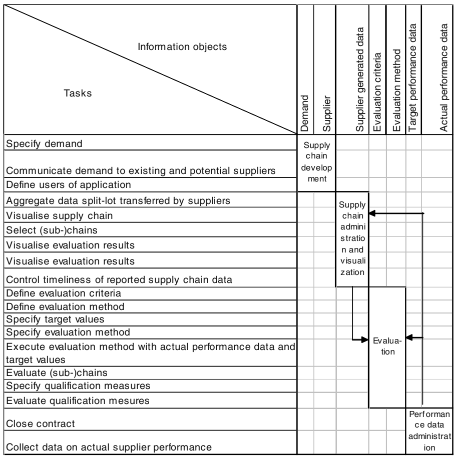
The business tasks are gained from the functional-decomposition diagram (Fig. 3) and are listed left on the table. Relevant groups of data are gained from the data model (Fig. 4) and utilised for the BCI process as information objects, listed on top. Such information objects are demand, supplier, supplier generated data, evaluation criteria, evaluation method, target performance data and actual performance data.

Table 2. Grouping of tasks and information objects

Tasks \ Information objects	Demand	Supplier	Supplier generated data	Evaluation criteria	Evaluation method	Target performance data	Actual performance data
Specify demand	C						
Communicate demand to existing and potential suppliers	U	U					
Define users of application		C					
Aggregate data split-lot transferred by suppliers			U				
Visualise supply chain			U				
Select (sub-)chains			U				
Visualise evaluation results			U			U	U
Visualise evaluation results			U				
Control timeliness of reported supply chain data			U				
Define evaluation criteria				C			
Define evaluation method					C		
Specify target values				C			
Specify evaluation method					C		
Execute evaluation method with actual performance data and target values				U	U	U	U
Evaluate (sub-)chains			U	U	U		
Specify qualification measures				C	C		
Evaluate qualification mesures				U	U	U	U
Close contract						C	
Collect data on actual supplier performance							C

An example relationship would be the usage "U" of supplier generated data for the visualisation of the supply chain. Four areas result for the domain of strategic supply chain development in changing the order of tasks and information objects in the matrix as defined in chapter 3. The four areas are potential business components. The first business component offers services for the specification of the demand and for the definition of application users. The component therefore provides services for the *supply chain development*. The second business component is responsible for the *supply chain administration and visualisation* in aggregating and managing the data received and in providing visualisation services. The *evaluation* component provides services in the area of evaluation methods and criteria, and the component on the right is responsible for the *performance data administration.* The "U"'s outside the grouped areas are substitute by arrows defining the dependencies of the single business components, shown in Table 3.

Table 3. Business components identified using the BCI method



Tasks \ Information objects	Demand	Supplier	Supplier generated data	Evaluation criteria	Evaluation method	Target performance data	Actual performance data
Specify demand	Supply chain development						
Communicate demand to existing and potential suppliers							
Define users of application							
Aggregate data split-lot transferred by suppliers			Supply chain administration and visualization				
Visualise supply chain							
Select (sub-)chains							
Visualise evaluation results							
Visualise evaluation results							
Control timeliness of reported supply chain data							
Define evaluation criteria				Evalua-tion			
Define evaluation method							
Specify target values							
Specify evaluation method							
Execute evaluation method with actual performance data and target values							
Evaluate (sub-)chains							
Specify qualification measures							
Evaluate qualification mesures							
Close contract						Performance data administration	
Collect data on actual supplier performance							

4.3 Refinement and Standard Specification of Business Components

The business component *supply chain administration and visualisation* gained from the BCI method needs to be partitioned in two components, the *supply chain administration* and the *visualisation and selection* component, in order to separate the administration of data from the visualisation of data. The separation from data and presentation is common within software development e.g. by patterns as Model View Control (MVC) [9] and can be used in a more abstract way also for business component development.

Two additional components are added to the business component model, namely the *communication manager* and the *communication*. These ones are attained from infrastructure requirements and not from business requirements and belong to the component system framework. Therefore they are not resulting from the BCI method.

The business component model gained from the BCI method and refined as just mentioned is shown in Fig. 5. For each component listed a specification for all levels of abstraction – marketing, task, terminology, quality, coordination, behavioural and interface – as defined in section 2, needs to be given. The description of the detailed specification of all levels for each single component is not within the scope of this paper.

Interesting for this paper instead is the specification of the services provided by the identified business components, defined in the task level of the specification memorandum. Single business tasks need therefore to be assigned to component services. The mapping for the strategic supply chain development domain is shown in Table 4 and is defined according the memorandum of standardised specification of business components.

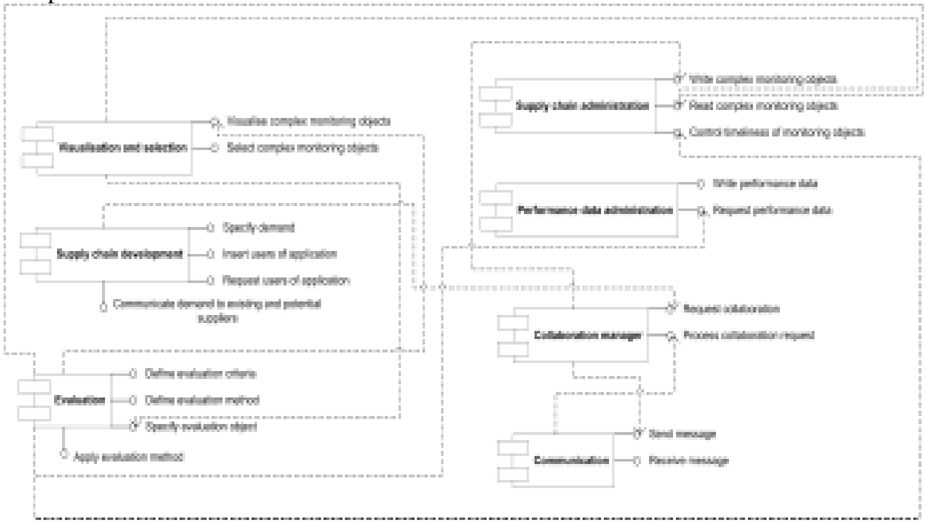

Supply chain administration
Write complex monitoring objects
Read complex monitoring objects
Control timeliness of monitoring objects
Visualisation and selection
Visualise complex monitoring objects
Select complex monitoring objects
Performance data administration
Write performance data
Request performance data
Supply chain development
Specify demand
Insert users of application
Request users of application
Communicate demand to existing and potential suppliers
Collaboration manager
Request collaboration
Process collaboration request
Evaluation
Define evaluation criteria
Define evaluation method
Specify evaluation object
Apply evaluation method
Communication
Send message
Receive message

Fig. 5. Refined business component model

Some business tasks are mapped one to one to business component services, e.g. *specify demand* or *communicate demand to existing and potential suppliers*. However most of the business tasks are mapped to a superordinate more abstract defined service, e.g. the business tasks *visualise supply chain* and *visualise evaluation results* are mapped to the service *visualise complex monitoring objects*. Fig. 5 shows the refined business component model with all services defined for each component and with the dependencies between the individual business components represented by arrows.

With the definition of a very abstract data model for the strategic supply chain development domain, as given in Fig. 4, powerful, high quality, reusable and self-contained business components can be identified which can be used with minimal integration effort and different parameterisations for different application systems.

Table 4. Mapping of business tasks to component services

Task	Service
Define users of application	Insert users of application
Specify demand	Specify demand
Communicate demand to existing and potential suppliers	Communicate demand to existing and potential suppliers
Aggregate data split-lot transferred by suppliers	Process collaboration request
Visualise supply chain(s)	Visualise complex monitoring objects
Define evaluation criteria	Define evaluation criteria
Define evaluation method	Define evaluation method
Select (sub-)chains	Selection of complex monitoring objects
Control timeliness of reported supply chain data	Control timeliness of monitoring objects
Evaluate (sub-)chains	Apply evaluation method
Visualise evaluation results	Visualise complex monitoring objects
Identify target suppliers for negotiations	*(execute manually)*
Conduct contract negotiations	*(execute manually)*
Specify contract termn	*(execute manually)*
Close contract	Write performance data
Collect data on actual supplier performance	Write performance data
Specify target values	Define evaluation criteria
Specify evaluation method	Define evaluation method
Execute evaluation mehtod with actual performance data and target valu	Apply evaluation method
Visualise evaluation results	Visualise complex monitoring objects
Specify qualification measures	Write performance data
Agree on qualification measures with suppliers	*(execute manually)*
Execute qualification measures	*(execute manually)*
Evaluate qualification measures	Apply evaluation method

5 Component Based Domain Design

The goal of the component based domain design is to elaborate different diagrams providing more information about instantiations and collaboration of the different components. Therefore two types of diagrams are appropriate, a deployment and a sequence diagram. Both are presented in the following subsection using the Unified Modelling Language [17 p. 362 and 432] notation.

5.1 Business Components Collaboration Design

For the domain of supply chain management different systems exist where the components are used, namely the producer system and all suppliers systems. Fig. 6 shows the deployment of business component instances on these systems.

Three systems are shown, one producer and two supplier systems. The producer holds all components belonging to the strategic supply chain development system including the supply chain development and the evaluation components. The suppliers utilise those components necessary for recording and sending the own data to the producer and for acquiring the data from their own suppliers. For a more coherent display not all component services and dependencies are shown. The dependencies and services calls are presented by arrows. The dependencies and the dataflow are given by means of an example and are illustrated in Fig. 7.

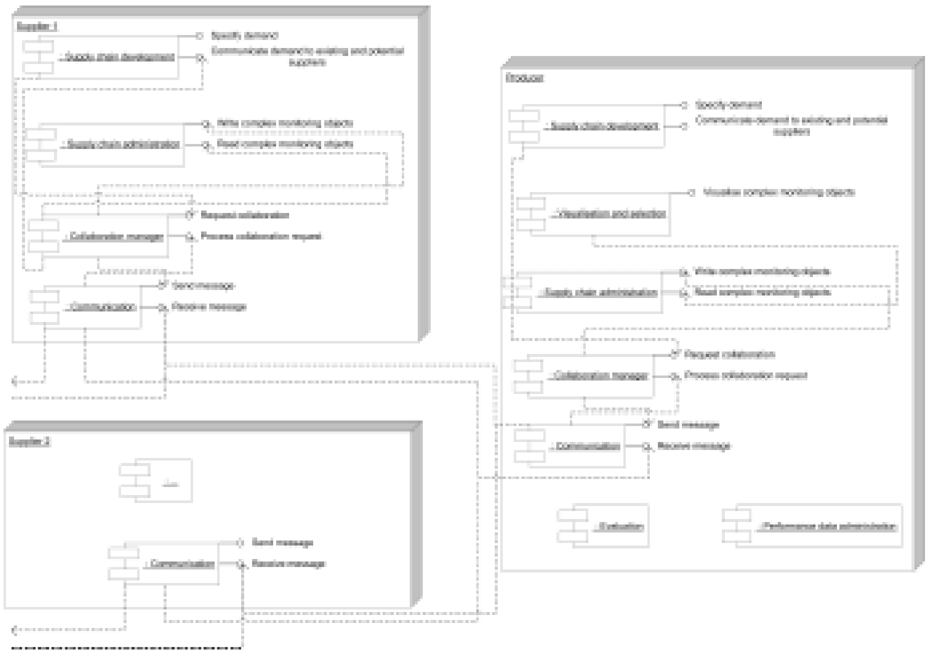

Supplier 1
: Supply chain development
Specify demand
Communicate demand to existing and potential suppliers
: Supply chain administration
Write complex monitoring objects
Read complex monitoring objects
: Collaboration manager
Request collaboration
Process collaboration request
: Communication
Send message
Receive message
Producer
: Supply chain development
Specify demand
Communicate demand to existing and potential suppliers
: Visualisation and selection
Visualise complex monitoring objects
: Supply chain administration
Write complex monitoring objects
Read complex monitoring objects
: Collaboration manager
Request collaboration
Process collaboration request
: Communication
Send message
Receive message
: Evaluation
: Performance data administration
Supplier 2
: ...
: Communication
Send message
Receive message

Fig. 6. Instantiation and deployment diagram

For the strategic development of a supply chain, defined by a demand, it is essential to collect complete data not only from the supplier in tier-1 but also from all suppliers in tier-n in order to be able to model and evaluate existing and alternative supply chains. Using the services of the supply chain development component a producer is able to specify the demand and to communicate the demand to all suppliers in tier-n. Triggered by that request, the supply chain development component accesses the service *request collaboration* of the collaboration manager component which uses the *send message* service of the communication component in order to send the demand to the suppliers in tier-1. The service requests of the different components are visualised in Fig. 7 by arrows. The communication components of the suppliers in tier-1 receive the message sent by the producer and forward the request to their collaboration managers accessing the *process collaboration request* service. Each collaboration manager uses the *communicate demand to existing and potential suppliers* service of the supply chain development component to forward the demand to existing and potential suppliers. The collaboration manager and the communication components are responsible for communication between the systems.

Supplier 1, having received the information data from supplier 3, stores the data using the service *write complex monitoring objects* in its own system. This information together with information about all the suppliers is sent back to the producer. At any time the producer receives the aggregated information from its suppliers in tier-1 and their suppliers. This information is given to the supply chain administration component in form of a complex monitoring object. Each user can then request to *visualise the complex monitoring object*. The complete supply chain is presented containing all information about the single supplier nodes. The producer is therefore able to evaluate and to develop the complete supply chain according to its requirements.

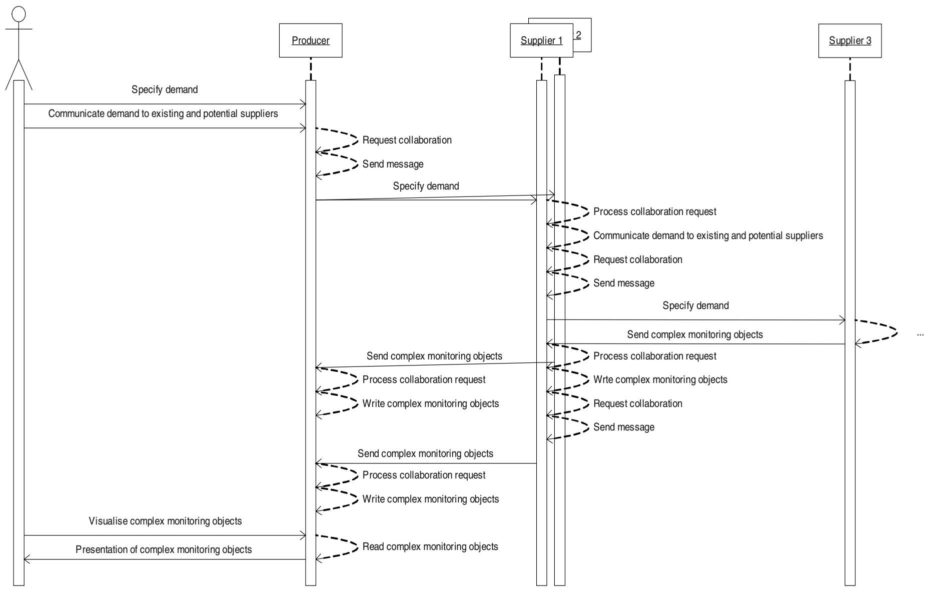

Producer
Supplier 1
2
Supplier 3
Specify demand
Communicate demand to existing and potential suppliers
Request collaboration
Send message
Specify demand
Process collaboration request
Communicate demand to existing and potential suppliers
Request collaboration
Send message
Specify demand
Send complex monitoring objects
...
Send complex monitoring objects
Process collaboration request
Process collaboration request
Wrte complex monitoring objects
Write complex monitoring objects
Request collaboration
Send message
Send complex monitoring objects
Process collaboration request
Write complex monitoring objects
Visualise complex monitoring objects
Presentation of complex monitoring objects
Read complex monitoring objects

Fig. 7. Sequence diagram

6 Conclusions

Having illustrated the necessity of building advanced information system using business components, a detailed explanation of business components has been given, explaining all seven contract levels, which need to be defined in order to specify a business component in a standard way. The identification of domain based business components which are reusable, self-contained and marketable is not as intuitive. Therefore the Business Components Modelling process has been introduced, explaining the single phases, sub phases and the resulting diagrams used in order to achieve a maximum result. With the BCM method high quality components can be identified and reused in different systems by different parameterisations and with view integration effort.

The BCM method has been verified for the domain of strategic supply chain development, which extends the traditional frame of reference in strategic sourcing from a supplier-centric to a supply-chain-scope including the dynamic modelling of strategic supply chain. All phases and sub phases for that domain have been detailed and the resulting diagrams have been explained.

Further work for the validation and refinement of the proposed methodology is required. In particular additional business domains need to be examined to obtain a broader assessment base. Additionally, the BCI process needs to be extended for the identification of infrastructural business components.

References

[1] *Business Systems Planning-Information Systems Planning Guide*, International Business Machines, Atlanta, 1984.

[2] D. Ammer, *Materials Management*, Homewood, 1968.

[3] C. Y. Baldwin and K. Clark, *Design Rules: The Power of Modularity*, MIT Press, Cambridge (Mass.), London, 2000.

[4] C. Y. Baldwin and K. Clark, *Managing in an age of modularity*, Harvard Business Review, 75 5 (1997), pp. 84–93.

[5] A. Beugnard, J.-M. Jézéquel, N. Plouzeau and D. Watkins, *Making Components Contract Aware*, IEEE Computer, 32 (1999), pp. 38–44.

[6] A. S. Carr, Pearson, J.N., *Strategically managed buyer - supplier relationships and performance outcomes*, Journal of Operations Management, 17 (1999), pp. 497–519.

[7] K. Czarnecki and U. Eisenecker, *Generative programming : methods, tools, and applications*, Addison Wesley, Boston, 2000.

[8] K. Fellner and K. Turowski, *Classification Framework for Business Components*, in R. H. Sprague, ed., *Proceedings of the 33rd Annual Hawaii International Conference On System Sciences*, IEEE, Maui, Hawaii, 2000.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley, Reading, Mass., 1995.

[10] H. Hahn and K. Turowski, *General Existence of Component Markets*, in R. Trappl, ed., *Sixteenth European meeting on Cybernetics and Systems Research (EMCSR)*, Vienna, 2002, pp. 105–110.

[11] K. C. Kang, *Feature-Oriented Domain Analysis (FODA) feasibility study*, Carnegie Mellon University Software Engineering Institute, Pittsburgh, Pa., 1990.

[12] L. Kaufmann, *Purchasing and Supply Management - A Conceptual Framework*, *Handbuch Industrielles Beschaffungsmanagement*, Hahn, D, Kaufmann, L. (Hrsg.), Wiesbaden, 2002, pp. 3–33.

[13] D. M. Lambert and M. C. Cooper, *Issues in Supply Chain Management*, Industrial Marketing Management, 29 (2000), pp. 65–83.

[14] T. W. Malone, Lautbacher, R.J, *The Dawn of the E-Lance Economy*, Harvard Business Review (1998), pp. 145–152.

[15] M. D. McIlroy, *Mass Produced Software Components*, in P. Naur and B. Randell, eds., *Software Engineering: Report on a Conference by the NATO Science Committee*, NATO Scientific Affairs Devision, Brussels, 1986, pp. 138–150.

[16] R. McIvor, Humphreys, P., McAleer, E., *The Evolution of the Purchasing Function*, Journal of Strategic Change, Vol. 6 (1997), pp. 165–179.

[17] OMG, *OMG Unified Modelling Language Spezification Version 1.4*, 2001.

[18] C. Rautenstrauch, H. Tangermann and K. Turowski, *Manufacturing Planning and Control Content Management in Virtual Enterprises Pursuing Mass Customization*, in C. Rautenstrauch, K. Turowski and R. Seelmann-Eggebert, eds., *Moving into Mass Customization - Information Systems and Management Principles*, Springer Verlag, Dubai, 2001, pp. 103–118.
[19] J. Sametinger, *Software engineering with reusable components*, Springer, Berlin ; New York, 1997.
[20] R. Sanchez, *Strategic flexibility in product competition*, Strategic Management Journal 16 (1995), pp. 135–159.
[21] R. Sanchez and J. T. Mahoney, *Modularity, flexibility and knowledge management in product and organization design*, Strategic management Journal, 17 (1996), pp. 63–76.
[22] M. Sawhney, Zabin, J., *The Seven Steps to Nirvana: Strategic Insights into eBusiness Transformation*, New York, 2001.
[23] M. A. Schilling, *Toward a general modular systems theory and its applications to interfirm product modularity*, Academy of Management Review, 25 (2000), pp. 312–334.
[24] A. Schmietendorf and A. Scholz, *Spezifikation der Performance-Eigenschaften von Softwarekomponenten*, *Modellierung und Spezifikation von Fachkomponenten - Workshop im Rahmen der MobIS 2000*, Siegen, 2000, pp. 41.
[25] A. Schmietendort and R. Dumke, *Spezifikation von Softwarekomponenten auf Qualitätsebenen*, *2. Workshop Modellierung und Spezifikation von Fachkomponenten (im Rahmen der vertIS 2001)*, Universität Bamberg, 2001, pp. 113–123.
[26] M. Simos, D. Creps, C. Klinger, L. Levine and D. Allemang, *Organization Domain Modeling (ODM) Guidebook*, 1996.
[27] C. Szyperski, *Component software: beyond object-oriented programming*, Addison-Wesley, Harlow, 1998.
[28] D. Tapscott, Ticoll, D., Lowy, A., *Digital Capital: Harnessing the Power of Business Webs*, Boston, 2000.
[29] K. Turowski, *Spezifikation und Standardisierung von Fachkomponenten*, Wirtschaftsinformatik, 43 (2001), pp. 269–281.
[30] K. Turowski, *Standardisierung von Fachkomponenten: Spezifikation und Objekte der Standardisierung*, in A. Heinzl, ed., *3. Meistersingertreffen*, Schloss Thurnau, 1999.
[31] K. Turowski, ed., *Vereinheitlichte Spezifikation von Fachkomponenten*, Arbeitskreis 5.3.10 der Gesellschaft für Informatik, Augsburg, 2002.
[32] K. T. Ulrich, *The role of product architecture in the manufacturing firm*, Research Policy, 24 (1995), pp. 419–440.
[33] H.-J. Warnecke, *Vom Fraktal zum Produktionsnetzwerk. Unternehmenskooperationen erfolgreich gestalten*, Berlin, 1999.

An Optimal Divide-Conquer Algorithm for 2D Skyline Queries

Hai-Xin Lu, Yi Luo, and Xuemin Lin

School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
{cshlu,luoyi,lxue}@cse.unsw.edu.au
http://www.cse.unsw.edu.au

Abstract. Skyline query processing is fundamental to many applications including multi-criteria decision making. In this paper, we will present an optimal algorithm for computing skyline in the two dimensional space. The algorithm has the progressive nature and adopts the divide-conquer paradigm. It can be shown that our algorithm achieves the minimum I/O costs, and is more efficient and scalable than the existing techniques. The experiment results demonstrated that our algorithm greatly improves the performance of the existing techniques.

1 Introduction

Given a set of points, "skyline query" returns the points in the set, which are not "dominated" by another point (see Fig. 1 for example). Skyline query processing is fundamental to many applications, including multi-criteria decision making [12] where the optimal criteria are sometimes conflicting. For instance, consider the query "finding a cheap car with a recent model". This is a typical example involving the two conflicting goals, cheap price and recent model, since cars recently made are likely expensive. Instead of identifying an optimal answer to the query, it may only be possible to provide a set of candidate answers that likely meet the user requirements. Clearly, some users may be able to afford a car with the latest model, while others may only be able to choose cars as new as possible within their budgets. In this application, it is unlikely that one will consider cars both older and more expensive than another; thus, the set of candidate cars form skyline (see Fig. 1 for example).

Efficient main-memory algorithms for computing skyline were first investigated in [6]. To manage large datasets which may not necessarily fit in main-memory, two skyline algorithms were developed in [1]. The algorithm *Divide-Conquer* [1] proposed to divide a data space into partitions so that main memory algorithms can be applied to each partition, while *Block-Nested-Loop* proposed to maintain a candidate list for skyline. Both Divide-Conquer and Block-Nested-Loop require multiple reads of a dataset. Further, they are not suitable for online processing of skyline queries since the first skyline point cannot be generated until the entire dataset is scanned. To resolve these problems, two new algorithms *Bitmap* and *Index* were developed in [13].

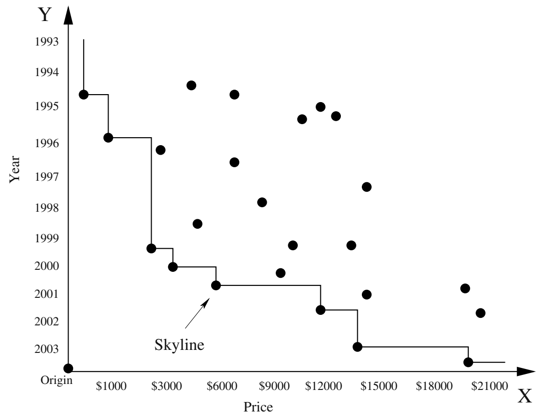


Fig. 1. An example of Skyline of cars

Note that the above algorithms take the same assumption - there is no spatial index. The algorithm (NN) [7] is the first algorithm for computing skyline based on R-trees [2,4], which uses the nearest neighbour search techniques [5,11]. Observe [10] the heavy I/O overheads in NN especially in a high dimensional space. The algorithm (BBS) [10] was developed to minimise the I/O costs, which extended the best-first search technique [5] for the nearest neighbour problem. As reported in [10], BBS guarantees the minimum I/O overheads but requires a larger memory space than that in NN in the two dimensional space. Note that less the memory space requires, more scalable the algorithm is.

In this paper, we will present a novel algorithm for computing skyline in the two dimensional space. The algorithm guarantees the minimum I/O costs, and requires a smaller memory space than the existing techniques (thus is more scalable than the existing techniques). Further, our new algorithm does not require a global validation of those progressively generated local skyline points. Our experiment results demonstrated that the new algorithm not only shares the different advantages from the existing techniques but also is much more efficient than the existing techniques.

The rest of the paper is organised as follows. Next section is dedicated to some background knowledge and related work. In section 3, we present our new skyline algorithm and its performance analysis. Section 4 reports the experimental results. This will be followed by conclusion and remarks.

2 Preliminary

This section provides a brief overview of NN and BBS. To the best of our knowledge, NN and BBS are the only R-tree based algorithms for computing skyline.

Given a set of points S in the 2 dimensional space, a point p in S is a *skyline point* of S if p is not *dominated* by another point in S. A point (x', y') *dominates* another point (x, y) if $x' \leq x$ and $y' \leq y$. The *skyline* of S is the set of skyline points of S.

An *entry* in R-tree refers to a node's MBR or a data point. As depicted in Fig. 3, each e_i (for $0 \leq i \leq 11$) is an entry, as well as the data points from a to t. An entry is dominated by a point p if the *lower-left corner* of the entry is dominated by p.

Both NN and BBS apply R-trees based nearest neighbour search techniques [5,11]. Note that a nearest neighbour search returns the points closest to a given query point. NN and BBS need to deal with the *distance* from a rectangle (entry) to a point, that is, the minimum distance between p and a point on the rectangle. In this paper, we denote the distance from an entry to the query point by $minDist$.

2.1 Nearest Neighbours (NN)

Kossmann et al. [7] proposed a divide-conquer algorithm NN based on multiple calls of the nearest neighbour search algorithms [5,11]. The algorithm iteratively generates a nearest neighbour point to the origin in a given region, and is applicable to any *monotonic* distance function (for example, Euclidean distance). The basic idea is to enforce the nearest neighbour points generated iteratively to be part of skyline.

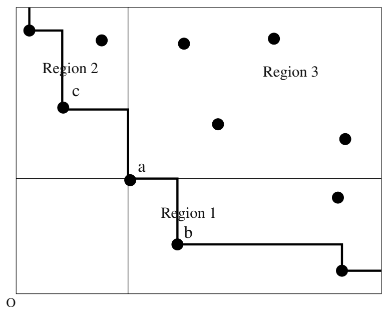


Fig. 2. Nearest Neighbours algorithm

As depicted in Fig. 2, the regions 1 and 2 are added to a *to-do* list in the algorithm for further space partitioning after a is output as the nearest neighbour point. Note that the points in region 3 are dominated by a, and thus can be discarded by the algorithm. The algorithm terminates when the to-do list becomes empty. To reduce the storage space in the to-do list, each region is simply recorded by its upper-right corner since the lower-left region resulted in a space devision is always empty.

Fig. 4 illustrates, step by step, the to-do lists of the algorithm NN based on the example in Fig. 3. In this example, b, k, h, and i form the skyline.

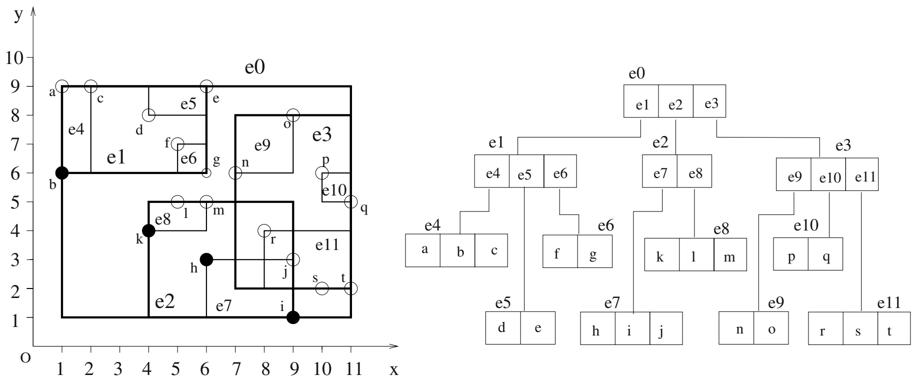


Fig. 3. R-tree example

Recall that in the to-do list every element corresponds to the coordinates of upper-right corner of a region. Note that in NN, a region is removed from the to-do list only if it is processed.

Action	**To-do List Contents**	**Skyline Points**	**To-do List Size**
initial state	$(11,9)$	$\emptyset$	1
NN query in $(11,9)$	$(11,6)$,$(1,9)$	$\{b\}$	2
NN query in $(11,6)$	$(11,4)$,$(4,6)$,$(1,9)$	$\{b,k\}$	3
NN query in $(11,4)$	$(11,3)$,$(6,4)$,$(4,6)$,$(1,9)$	$\{b,k,h\}$	4
NN query in $(11,3)$	$(11,1)$,$(9,3)$,$(6,4)$,$(4,6)$,$(1,9)$	$\{b,k,h,i\}$	5
NN query in $(11,1)$	~~$(11,1)$~~,$(9,3)$,$(6,4)$,$(4,6)$,$(1,9)$	$\{b,k,h,i\}$	5
NN query in $(9,3)$	~~$(9,3)$~~,$(6,4)$,$(4,6)$,$(1,9)$	$\{b,k,h,i\}$	4
NN query in $(6,4)$	~~$(6,4)$~~,$(4,6)$,$(1,9)$	$\{b,k,h,i\}$	3
NN query in $(4,6)$	~~$(4,6)$~~,$(1,9)$	$\{b,k,h,i\}$	2
NN query in $(1,9)$	~~$(1,9)$~~	$\{b,k,h,i\}$	1

Fig. 4. To-do List Contents

2.2 Branch and Bound Skyline Algorithm (BBS)

As pointed out in [10], multiple applications of a nearest neighbour search algorithm in NN may lead to multiple scans of entries in the R-tree. For instance, regarding the example in Fig. 3, to get the skyline points b and k both the root entry and the intermediate entry e_2 have to be visited (expanded). Further, in NN empty regions may be unnecessarily kept in the to-do list for long time. Moreover, NN leads to many duplicated elements in the to-do list and causes many *false* hits in a high dimensional space.

To resolve the problems in NN, the algorithm BBS developed by Papadias et al. extended the best-first search technique [5] by keeping all the necessary entries in a "heap" until they are no longer useful. Elements (entries) in the heap are sorted by their *minDist* values (to the origin), and then the skyline points are generated iteratively as the nearest neighbour points to the origin. For a dominance validation, the skyline points generated already are kept in a list T as BBS proceeds:

Algorithm BBS

```
Insert the root entry of R-tree into the heap $H$.
**While** $H \neq \emptyset$ **do**
{ get the first element $e$ from $H$;
  $H := H - \{e\}$;
  **If** $e$ is dominated by one point in $T$ **then** discard $e$
  **else if** $e$ is a data point **then** $T := T \cup \{e\}$
        **else** add to $H$ the children entries of $e$
              which are not dominated by a point in $T$; }
```

Fig. 5 shows the heap contents for the example in Fig. 3 while applying BBS. The second field in an element is a value of minDist where $x + y$ is the distance function.

Action	**Heap Contents**	**Skyline Points**	**Heap Size**
initial state	$(e_0, 2)$	$\emptyset$	1
expand e_0	$(e_2, 5)$,$(e_1, 7)$,$(e_3, 9)$	$\emptyset$	3
expand e_2	$(e_7, 7)$,$(e_1, 7)$,$(e_8, 8)$,$(e_3, 9)$	$\emptyset$	4
expand e_7	$(e_1, 7)$,$(e_8, 8)$,$(h, 9)$,$(e_3, 9)$,$(i, 10)$,$(j, 12)$	$\emptyset$	6
expand e_1	$(e_4, 7)$,$(e_8, 8)$,$(h, 9)$,$(e_3, 9)$,$(i, 10)$, $(e_6, 11)$,$(e_5, 12)$,$(j, 12)$	$\emptyset$	8
expand e_4	$(b, 7)$,$(e_8, 8)$,$(h, 9)$,$(e_3, 9)$,$(a, 10)$, $(i, 10)$,$(c, 11)$,$(e_6, 11)$,$(e_5, 12)$,$(j, 12)$	{b}	10
expand e_8	$(k, 8)$,$(h, 9)$,$(e_3, 9)$,$(l, 10)$,$(a, 10)$, $(i, 10)$,$(m, 11)$,$(c, 11)$,$(e_6, 11)$,$(e_5, 12)$,$(j, 12)$	{b,k,h}	11
expand e_3	$(e_{11}, 10)$,$(l, 10)$,$(a, 10)$,$(i, 10)$,$(m, 11)$, $(c, 11)$,$(e_6, 11)$,$(e_5, 12)$,$(j, 12)$	{b,k,h}	9
expand e_{11}	~~$(l, 10)$~~,~~$(a, 10)$~~,$(i, 10)$,~~$(m, 11)$~~, ~~$(c, 11)$~~, ~~$(e_6, 11)$~~,~~$(s, 12)$~~,~~$(e_5, 12)$~~,~~$(j, 12)$~~,~~$(t, 13)$~~	{b,k,h,i}	10

Fig. 5. Heap Contents

It has been shown that BBS is optimal in terms of I/O; that is, only "necessary" entries in the R-tree are visited. We will formally state the I/O optimality in the next section.

3 A New Divide-Conquer Algorithm

In this section, we present a new skyline algorithm based on a divide and conquer paradigm. The algorithm is more efficient than BBS and NN. It has a low memory space requirement and guarantees the I/O optimality. We start with a description of our motivation.

3.1 Motivation

Note that BBS achieves the I/O optimality by reading the same entries only once. This means that once reading into the buffer, the information has to be kept there till it is no longer useful. By doing this, the maximal size of the heap will be unnecessarily large if entries are not pruned early nor the R-tree is drilled down quickly.

Regarding the example in Fig. 3, Fig. 5 showed that in BBS, entries e_5 and e_6 have to be kept in the heap for quite long time after e_2 is expanded. However, as depicted in Fig. 6 we can use e_2 (even before expanding e_2) to prune e_5 and e_6. This additional pruning step can be formally done by specifying a dominance relationship between two entries (not necessary data points). An entry e is *dominated* by another entry e' if e is outside e' and is dominated by the lower-left corner of e'. In this example, e_5 and e_6 are dominated by e_2, and thus should be discarded from a further consideration.

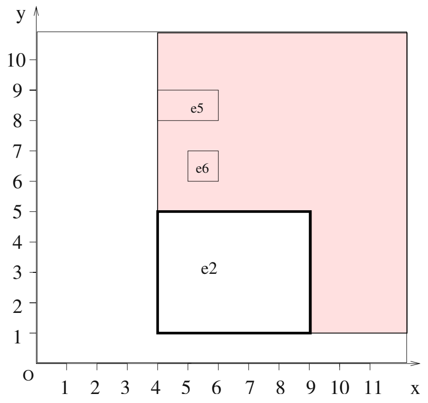


Fig. 6. Region Dominance

Fig. 5 also demonstrated that to generate b, BBS has to traverse two paths on the R-tree:

path 1: e_0, e_2, and e_7.
path 2: e_0, e_1, and e_4.

In this case, it will be ideal if only the path (e_0, e_1, e_4) is visited to generate b as this minimises the size of information concurrently held in the buffer. Our new algorithm can lead such an ideal situation in most cases.

Further, BBS maintains a list T of the current skyline points to validate, by a binary search on T, if a newly generated entry should be included for a further consideration; this expensive requirement is no longer needed in our algorithm. In fact, the new algorithm is I/O optimal (like BBS) and requires a much smaller buffer size than that in BBS. Moreover, our algorithm is much faster than BBS and NN.

3.2 DCSkyline

In this subsection, we present a novel divide-conquer skyline algorithm - DC-Skyline. The divide-conquer in DCSkyline proceeds in a different way from that in NN.

Where to Divide. DCSkyline traverses the R-tree level by level like a depth-first search paradigm. After expanding an intermediate entry, it always chooses the entry in a region, with the minimum distance to the origin, to divide the region.

For instance, regarding the example in Fig. 7, e_2 is first chosen if the distance function $x+y$ is used. Then, DCSkyline uses the lower-left corner of e_2 to divide the whole space into 4 rectangular regions for further consideration, R_0, R_1, R_2, and R_3 (e_2), as depicted in Fig. 7. Note that entries outside these 4 regions are dominated by e_2 and thus should be discarded. Since the distance from e_2 to the origin is minimum, R_0 is always empty (i.e. no intersection with the entries in the R-tree). Therefore, in our algorithm we need only to consider the three regions R_1, R_2, and R_3. Below we show the properties in these three regions, which will lead to an effective divide-conquer process in DCSkyline.

Region Properties. Suppose that in a rectangular region R, the entry e has the minimum distance to the origin. On the dividing point (the lower-left corner of e), R is divided into the 4 rectangular regions for further consideration: R_0, R_1, R_2, and R_3, as depicted in Fig. 7.

Theorem 1. *The local skyline points in R_1 or R_2 are skyline points in R.*

Proof: Denote the dividing point by (x, y). Without loss of generality, we assume the lower-left corner of R is the origin, and $R = [0, \infty) \times [0, \infty)$. Then, $R_1 = [0, x) \times [y, \infty)$, $R_2 = [x, \infty) \times [0, y)$, and $R_3 = [x, x_h] \times [y, y_h]$ where (x_h, y_h) is the upper-right corner of e.

Without loss of generality, we prove only that the theorem holds for region R_1. Note that $x_1 < x$ for each point $(x_1, y_1) \in R_1$, $x_2 > x$ for each point $(x_2, y_2) \in R_2$, and $x_3 \geq x$ for each point $(x_3, y_3) \in R_3$.

Since $x_1 < x_2$, (x_1, y_1) is not dominated by (x_2, y_2). Similarly, (x_1, y_1) is not dominated by (x_3, y_3). Thus the theorem holds for R_1. $\Box$

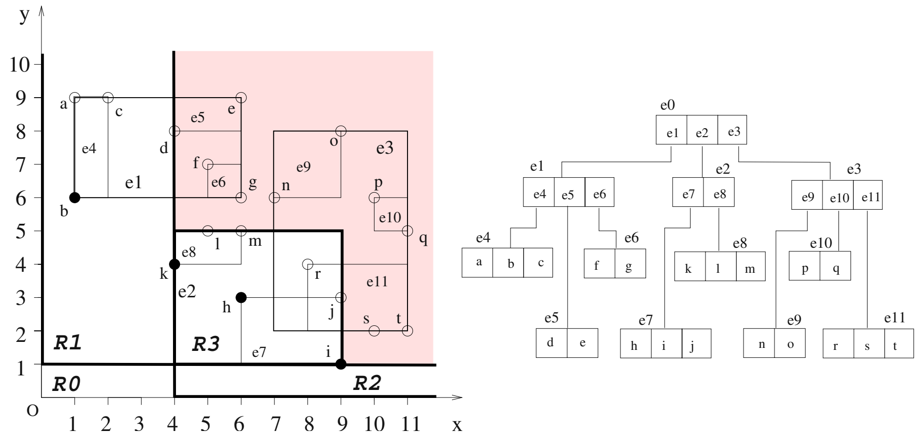


Fig. 7. Example of DCSkyline

Theorem 1 states that the local skyline points generated in R_1 and R_2, respectively, do not have to be verified globally. Clearly, the set of local skyline points generated in R_1 do not intersect the set of local skyline points in R_2.

Unfortunately, some local skyline points generated in the region R_3 may not be skyline points in R. To avoid a global check, we can use the right-most skyline point in R_1 and the left-most skyline point in R_2 to further restrict our search in the region R_3.

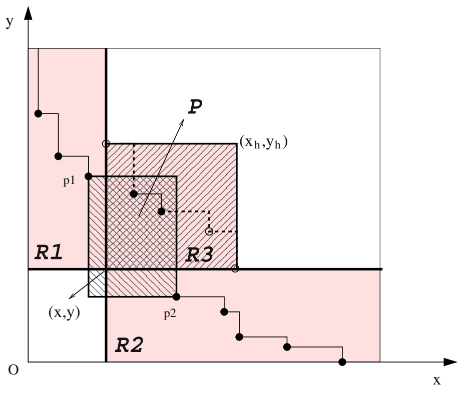


Fig. 8. The Principle of D&C

Let p_1 denote the right-most skyline point in R_1, p_2 denote the left-most skyline point in R_2, and P denote the rectangle spanning p_1 and p_2. Let $R'_3 = R_3 \cap P$; see Fig. 8 for example. Note that $p_1 = (0, \infty)$ if there is no skyline point in R_1; and $p_2 = (\infty, 0)$ if there is no skyline point in R_2. The following theorem can be immediately verified.

Theorem 2. *The local skyline points in R'_3 are skyline points in R. Further, the skyline of R consists of the local skyline points from R_1, R_2, and R'_3.*

Based on Theorems 1 and 2, our algorithm DCSkyline recursively applies a divide-conquer paradigm by the following ordering:

1. Find the skyline in R_1 by calling DCSkyline.
2. Find the skyline in R_2 by calling DCSkyline.
3. Use p_1 and p_2 to get R'_3, and call DCSkyline on R'_3 to get the skyline.

Region List in DCSkyline. In DCSkyline, we use the *region-list* to store the rectangular regions to be further divided; only regions that may contain skyline points will be pushed into the region list. In the example in Fig. 7, R_2 is empty, and thus is not in the list.

In the region-list, we also store the expanded entries associated with each region; that is, for each region, we store the entries already visited in the R-tree that intersect the region.

Note that an entry may appear in more than one region. Consider the example in Fig. 9. In the first step when the whole space is divided by the lower-left corner of e_2, e_1 is contained in both R_1 and R_3. To guarantee to read e_1 only once and expand e_1 at most once, e_1 is linked to R_1 and R_3. Further, after expanding e_1, the children entries of e_1 may also be involved in R_1 and R_3; for example e_3 in Fig. 9. Therefore, if e_1 is expanded when processing R_1, we also need to check whether or not its children should be linked to R_3.

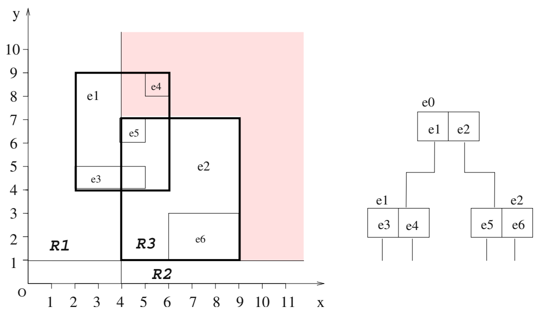


Fig. 9. Entries cross different regions

Note that R_1 and R_2 never intersect the same entry in the R-tree.

Expanding Entries. An entry is expanded if it is either the only entry intersecting R_1 (or R_2) or it is the entry corresponding to R_3. Note that in DCSkyline if the entry to be expanded is a data point, then it is output as a skyline point.

Pruning Regions and Entries.

- If an entry e is not in any region, then the entry is discarded.
- If an entry is expanded, then the entry is discarded.
- If a region is empty, then the region is discarded.

DCSkyline Algorithm. The pseudo-code of the algorithm is shown in Fig. 10.

```
Initialization.
$E$ := set of children entries of the root of $R$-tree;
$R$ := the data space;
$RT$ := $R$-tree;
$S := \emptyset$ // skyline points
Algorithm DCSkyline ($E$, $R$, $S$)
   if $R$ is a type 3 Region then** // expand an entry
     {remove $e$ (the entry defining the region $R$) from $E$
      as well as from the other intersecting regions;
      add the appropriate children entries of $e$ to $E$ and the other
      intersecting regions; }
   while $|E| = 1$ and the entry $e$ in $E$ is not a data point do**// expand an entry
     {remove $e$ from $E$ and the other intersecting regions;
      add the appropriate children entries of $e$ to $E$ and the other
      intersecting regions; }
   if $|E| = 1$ and the entry $e$ in $E$ is a data point then $S := \{e\}$
   else if $|E| = 0$ then $S := \emptyset$
        else // $|E| > 1$
        { choose the closest entry $e$ from $E$;
          divide $R$ into $R_1$, $R_2$ and $R_3$ using $e$ (as depicted in Fig. 7);
          $E_1 := E \cap R_1$; $E_2 := E \cap R_2$; $E_3 := E \cap R_3$; // entry lists
          $S_1 := \emptyset$; $S_2 := \emptyset$; $S_3 := \emptyset$; // skyline points
          if $|E_1| > 0$ then DCSkyline ( $E_1$ , $R_1$ , $S_1$);
          if $|E_2| > 0$ then DCSkyline ( $E_2$ , $R_2$ , $S_2$);
          let $p_1$ the right-most skyline point from $S_1$; //If $S_1 = \emptyset$ then $p_1 = (0, \infty)$
          let $p_2$ the left-most skyline point from $S_2$; //If $S_2 = \emptyset$ then $p_2 = (\infty, 0)$
          let $P$ the region spanning $p_1$ and $p_2$ (as depicted in Fig. 8);
          $R_3 := R_3 \cap P$; $E_3 := E_3 \cap P$;
          if $|E_3| > 0$ then DCSkyline ( $E_3$ , $R_3$ , $S_3$);
          $S := S_1 \cup S_2 \cup S_3$ }
end
```

$**$ As depicted in Fig. 9, an entry may appear in more than one region. To guarantee to read each entry only once, extending an entry should be carried out in all the current regions intersecting the entry. In our algorithm, we linked an entry to the intersecting regions for efficiency.

Fig. 10. Pseudo Code

Fig. 11 provides a sample run of the algorithm DCSkyline based on the example in Fig. 7, where only the region-list is illustrated. Note that every element in the region-list has two fields representing the region and the set of corresponding entries.

Action	**Region list Contents**	**Skyline Points**	**Region + Entry List Size**
initial state	$[R, \{(e_0, 2)\}]$	$\emptyset$	$1 + 1$
expand e_0 & divide R	$[R_1, \{(e_1, 7)\}]$,$[R_3, \{(e_2, 5), (e_3, 9)\}]$	$\emptyset$	$2 + 3$
expand e_1	$[R_1, \{(e_4, 7)\}]$,$[R_3, \{(e_2, 5), (e_3, 9)\}]$	$\emptyset$	$2 + 3$
expand e_4	~~$[R_1, \{(b, 7)\}]$~~,$[R_3, \{(e_2, 5), (e_3, 9)\}]$	{b}	$2 + 3$
expand e_2 & divide R_3	$[R_{31}, \{(e_8, 8)\}]$,$[R_{33}, \{(e_7, 7), (e_3, 9)\}]$	{b}	$2 + 3$
expand e_8	~~$[R_{31}, \{(k, 8)\}]$~~,$[R_{33}, \{(e_7, 7), (e_3, 9)\}]$	{b,k}	$2 + 3$
expand e_7 & divide R_{33}	~~$[R_{333}, \{(h, 9)\}]$~~,$[R_{332}, \{(e_3, 9), (i, 10)\}]$	{b,k,h}	$2 + 3$
divide R_{332}	~~$[R_{3322}, \{(i, 10)\}]$~~,$[R_{3323}, \{(e_3, 9)\}]$	{b,k,h,i}	$2 + 2$
expand e_3	$[R_{3323}, \{(e_{11}, 10)\}]$	{b,k,h,i}	$1 + 1$
expand e_{11}	$\emptyset$	{b,k,h,i}	$0 + 0$

Fig. 11. Region-List Contents

In Fig. 11, $R_{d_1..d_l i}$ $(i = 1, 2, 3)$ denotes the region with type R_i in the space division of $R_{d_1..d_l}$.

3.3 Analysis of DCSkyline

By the theorems 1 and 2, it immediately follows that DCSkyline is correct; that is, the points generated form the skyline of the dataset. Next, we prove that DCSkyline accesses the minimum number of entries in the R-tree. We first need the following concept.

Suppose that the skyline of a dataset S is given. The part of space that have to be searched is called *skyline search region* (SSR). The SSR is the area defined by the skyline and the two axes, as depicted by the shaded area in Fig. 12.

In [10], it has been shown that every correct skyline algorithm based on R-trees must access all the entries intersecting SSR. As enforced, DCSkyline access an entry only once. Therefore, to prove the optimality of I/O in DCSkyline we need only to prove the following theorem.

Theorem 3. *Suppose that e is an entry accessed by DCSkyline. Then e intersects SSR.*

Proof: As discussed in the last subsection, an entry e accessed by DCSkyline must satisfy one of the two conditions: 1) it is the only entry left in the current R_1 (or R_2), or 2) it is the entry corresponding to R_3. Note that R_1, R_2, and R_3 may not at the same level of a space division.

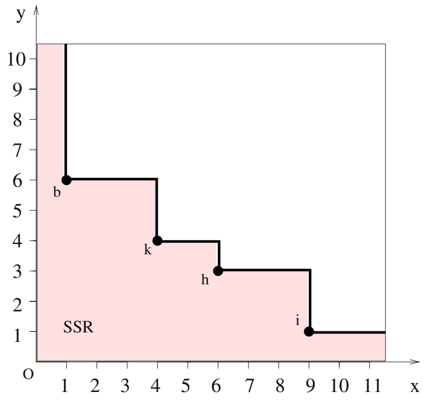


Fig. 12. SSR

Assume that on the contrary, e does not intersects SSR. There is only one possibility: e is dominated by a skyline point p. It can be immediately shown that one of the above two conditions for accessing e eliminates the possibility. □

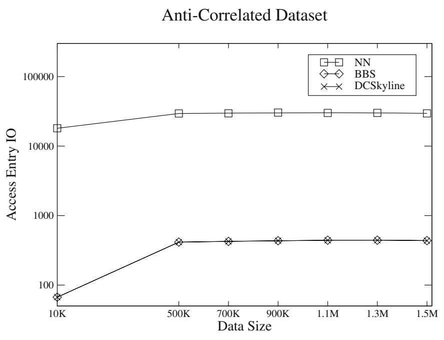
Clearly, the algorithm DCSkyline is *progressive*; that is, the skyline points are generated iteratively without processing the whole dataset. We may expect that DCSkyline is more efficient than BBS since BBS has to perform a global check every time when a candidate skyline point is generated and BBS has to process a large heap list many times. Further, we may expect that DCSkyline is more efficient than NN as NN has very high I/O overheads. These, together with a small buffer requirement, make our algorithm DCSkyline combine all the advantages in the existing techniques.

4 Experimental Results

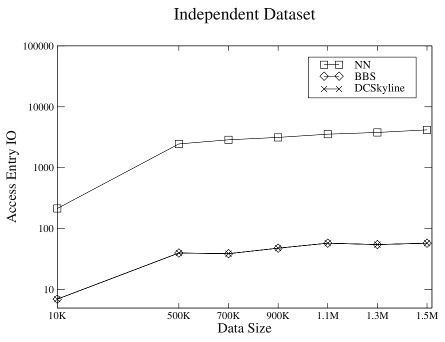
In this section, we evaluate the performance of DCSkyline. In our experiment, NN and BBS are used as the benchmark algorithms. *Independent* and *anti-correlated* [7] datasets (up to 1.5M points) are used and indexed by the $R*$-tree algorithm in [8] with node capacity of 200. Our experiment was carried out on a PENTIUM III/864 CPU with 256MB Ram.

I/O Costs. Fig. 13 illustrated the number of entries accessed in the three algorithms. Both BBS and DCSkyline accesses the minimum number of entries, which is much smaller than that accessed by NN.

Memory Utilisation. To evaluate the buffer requirements of NN, BBS, and DCSkyline, in our experiment we recorded the maximum number of elements in the to-do list of NN, in the heap of BBS, and in the region-list of DCSkyline,



(a) Anti-correlated Datasets

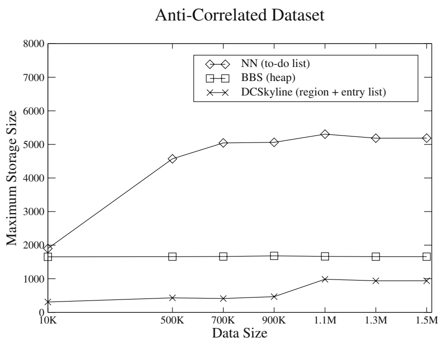


(b) Independent Datasets

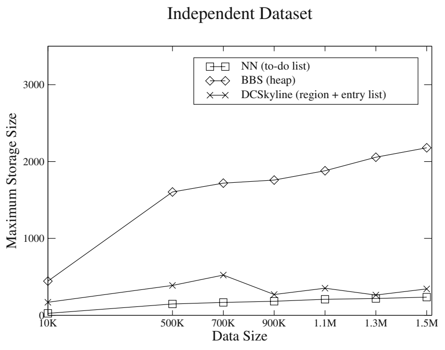
Fig. 13. Access Entry IO

respectively. Note that when we count the number of elements in the region-list we count a region and an entry, respectively, as an element. Our experiment results are reported in Fig. 14.

As clearly demonstrated, DCSkyline requires much smaller buffer space than that by BBS. It is interesting to note that DCSkyline requires slightly larger buffer spaces than those required by NN for independent datasets; however DC-Skyline clearly outperforms NN for anti-correlated datasets.



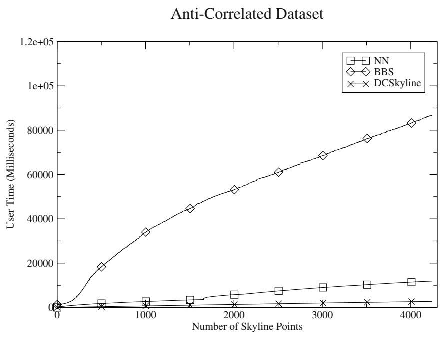
(a) Anti-correlated Datasets



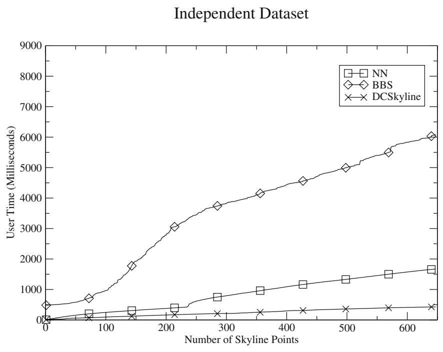
(b) Independent Datasets

Fig. 14. Maximum Storage Size

Efficiency. Fig. 15 demonstrates CPU time used in these algorithms as a function of the number of skyline points returned. DCSkyline improves the time efficiency of BBS by orders of magnitude, even with respect to the progressive behaviours. DCSkyline is also much faster than NN.



(a) Anti-correlated Datasets



(b) Independent Datasets

Fig. 15. User Time

In summary, our experiment results demonstrated that the new algorithm DCSkyline, presented in this paper, is more efficient and requires much less memory space than those in the existing algorithms. Further, our experiments also verified the one important property of DCSkyline - a guarantee of the minimum I/O costs as what is proved in the paper.

5 Conclusion and Future Works

In this paper, we presented a novel algorithm based on R-trees for computing the skyline of a dataset in the two dimensional space. Our new algorithm guarantees the minimum I/O costs, small buffer requirements, and more efficient than the existing techniques.

As a possible future study, we will explore the ways to apply our techniques developed in the paper to a high dimensional space by using effective variations of R-tree, such as X-tree.

References

1. S. Borzsonyi, D. Kossmann and K. Stocker. "The Skyline Operator", *International Conference on Data Engineering ICDE*, 2001.
2. N. Beckmann, H. Kriegel, R. Schneider, B. Seeger. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *SIGMOD*, 1990.
3. H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, A. Abbadi. "Constrained Nearest Neighbor Queries", *SSTD*, 2001.
4. V. Gaede and O. Gunther. "Multidimentional Access Methods", *Computing Surveys*, 30(2): 170–231, 1998.
5. G. Hjaltason, H. Samet. "Distance Browsing in Spatial Databases", *ACM TODS*, 24(2):265–318, 1999.
6. H.T. Kung, F. Luccio and F.P. Preparata. "On finding the maximum of a set of vectors", *Journal of the ACM, 22(4):469–476*, 1975.

7. D. Kossmann, F. Ramsak and S. Rost. "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries", *VLDB'02*, 2002.
8. Marios Hadjieleftheriou. "Spatial Index Library", http://www.cs.ucr.edu/ marioh/spatialindex/index.html, 2002.
9. F.P. Preparata and M.I. Shamos. "Computational Geometry: An Introduction", *Springer-Verlag, New York, Berlin, etc..*, 1985.
10. D. Papadias, Y. Tao, G. Fu, B. Seeger. "An Optimal Progressive Alogrithm for Skyline Queries", *SIGMOD*, 2003.
11. N. Roussopoulos, S. Kelly, F. Vincent. "Nearest Neighbor Queries", *SIGMOD*, 1995.
12. R. Steuer "Multiple Criteria Optimization", *Wiley New York*, 1986.
13. K.L. Tan, P.K. Eng and B.C. Ooi. "Efficient Progressive Skyline Computation", *VLDB*, 2001.

Hierarchical Data Cube for Range Queries and Dynamic Updates

Jianzhong Li[1,2] and Hong Gao[1,2]

[1] School of Computer Science and Technology,
Harbin Institute of Technology, Harbin, 150001, China
[2] School of Computer Science and Technology,
Heilongjiang University, Harbin, 150001, China
{lijz,gaohong}@mail.banner.com.cn

Abstract. A range query is a very popular and important operation on data cube in data warehouses. It performs an aggregation operation (e.g., SUM) over all selected cells of an OLAP data cube where the selection is specified by providing ranges of values for dimensions. Several techniques for range sum queries on data cubes have been introduced recently. However, they can incur update costs in the order of the size of the data cube. Since the size of a data cube is exponential in the number of its dimensions, updating the entire data cube can be very costly and not realistic. To solve this problem, an effective hierarchical data cube (HDC for short) is provided in this paper. The analytical and experimental results show that HDC is superior to other cubage storage structures for both dynamic updates and range queries.

1 Introduction

On-Line Analytical Processing [1] (OLAP) is a powerful tool in data warehouse to support decision- making. An increasingly popular data model for OLAP applications is the multidimensional database (MDDB), also known as a data cube [2]. To build a data cube for a data warehouse, certain attributes are chosen to be measure attributes, i.e., the attributes whose values are of interest, Other attributes, say *d* of them, are referred to as dimension attributes. Thus, a data cube can be viewed as a *d*-dimensional array, indexed by the values of the *d*-dimensional attributes, whose cells contain the values of the measure attributes for the corresponding combination of dimension attributes.

Range queries are useful analysis tools when applied to data cubes. A range query applies an aggregation operation (e.g., SUM) over all selected cells in a data cube. The cells are specified by the range query through providing ranges of values for dimensions. Supposing $S=[l_1, h_1]\times[l_2, h_2]\times\ldots\times[l_d, h_d]$ being a sub-region in the *d*-dimensional cube *C*, then the range query on *S* can be formulized as $f(S)=f(\{C(x_1, x_2, \ldots, x_d) \mid \forall(x_1, x_2, \ldots, x_d)\in S\})$, where *f* is an aggregate function, such as sum etc.

It is natural that the response time is crucial for OLAP application that needs user-interaction. The intuitive method to process the range query is to access the data cube itself. But it suffers from the fact that too many cells need to be accessed to get the aggregation. The number of cells to be accessed is proportional to the size of the sub-cube defined by the query. To enhance search efficiency, the prefix sum approach [3] has been proposed, which uses an additional cube called a prefix sum cube (PC), to store the cumulative sum of data. The prefix sum method permits the evaluation of a range query on a data cube in constant time. However, the approach is hampered by its update cost, which in the worst case requires rebuilding an array of the same size as the entire data cube. The current enterprise environment forces data elements in the cube to be dynamically changed. Managers demand that their analysis tools provide current or "near-current" information. In such an environment, the response time is affected by the update cost as well as the search cost of the cube.

Recently, various studies [3,4,5,6,7,8,9,10] have been made to reduce the update cost. However, those methods have some limitation in the context that they (1) still have the update propagation problem even though they have reduced it in some degree, such as the relative prefix sum cube (RPC), or (2) sacrifice the search efficiency to reduce the update cost. In many OLAP applications, it becomes an important issue to improve the update performance while minimizing the sacrifice of the search efficiency, and as well as the additional space.

In this paper, we propose a new cube storage structure, called hierarchical data cube (HDC for short). Compared to the related approaches proposed in the literatures, HDC has the following advantages: (1) it has the minimal time cost for both range sum queries and updates, which is O($\log^d n$), and (2) no more space overhead than other existed methods. The analytical and experimental results show that HDC is superior to other cube storage structures for updates and range queries.

The remainder of the paper is organized as follows. Section 2 introduces the model for the range sum problem and the optimization measure of the problem under two different (time) cost models. Some existed techniques are also reviewed. In Section 3, the hierarchical data cube HDC and the range query and update algorithms are proposed. Section 4 gives the performance analysis of HDC and the experimental results. Finally, the summary of this paper and the future work are given in Section 5.

2 Preliminaries

Assume that data cube C has one measure attribute and d-dimensional attributes, and is stored in a d-dimensional array A with size of $n_1 \times n_2 \times \ldots \times n_d$, where n_i is the size of ith dimensional attribute. Without loss of generality and for simplicity of analysis, we assume that A has a starting index 0 at each dimension, and each dimension has the same size, where $n_1 = n_2 = \ldots\ldots = n_d = n$. Note that this assumption is not a prerequisite to HDC.

In the prefix sum method (PS)[3], besides A, another d-dimensional array PC is introduced, which has the same size as array A. PC is used to store various pre-computed prefix sums of A. Each cell indexed by $(x_1, x_2, \ldots, x_d)$ in PC contains the sum of all cells up to and including itself in array *A*. Thus, the sum of the entire array *A* is found in the last cell of PC. In other words, we will pre-compute, for all $0 \le x_i \le n$ and $0 \le i \le d$, $PC(x_1, x_2, \ldots, x_d) = Sum(\{A(y_1, y_2, \ldots, y_d) \mid \forall 1 \le i \le d, 0 \le y_i \le x_i\})$. Fig. 1 illustrates the main idea of PS approach when d=2. From Fig. 1, we can get $PC(1,2)=A(0,0)+A(0,1)+A(0,2)+A(1,0)+A(1,1)+A(1,2)=21$.

	0	1	2	3	4	5	6	7
0	3	5	1	2	2	4	6	3
1	7	3	2	6	8	7	1	2
2	2	4	2	3	3	3	4	5
3	3	2	1	5	3	5	2	8
4	4	2	1	3	3	4	7	1
5	2	3	3	6	1	8	5	2
6	4	5	2	7	1	9	3	3
7	2	4	2	2	3	1	9	1

	0	1	2	3	4	5	6	7
0	3	8	9	11	13	17	23	26
1	10	18	21	29	39	50	57	62
2	12	24	29	40	53	67	78	88
3	15	29	35	51	67	86	99	117
4	19	35	42	61	80	103	123	142
5	21	40	50	75	95	126	151	172
6	25	49	61	93	114	154	182	206
7	27	55	69	103	127	168	205	230

Fig. 1. A 2-dimensional cube C(left)and its corresponding PC(right)

Fig. 2 presents the essential idea about how PC can be used to answer a range sum query. The sum of all the data items in the region selected by a range query can be computed by adding and subtracting the sums of various other regions until the interesting region is extracted. For example, sum(Area_E)=*PC*(4,4)-*PC*(4,1)-*PC*(1,4) +*PC*(1,1). We note that all such regions begin at cell A[0,0] and extend to some cell in *A*, thus, the prefix sum approach reduces the range sum query problem to the problem of reading a single individual cell in array *PC* that takes constant time.

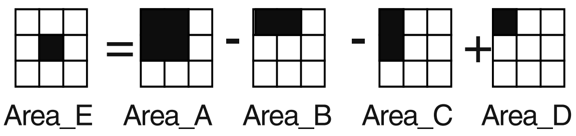


Fig. 2. A geometric illustration of the two dimensional case:Sum(Area_E)=Sum(Area_A)-Sum(Area_B)-Sum(Area_C)+Sum(Area_D)

The PS approach provides range sum queries in constant time, regardless of the size of the data cube. On the other hand, the update cost is very expensive. In the worst case, the update cost is $O(n^d)$.

To reduce the update propagation, Geffner et el.[4] presented a *relative prefix sum* technique, whose update requires $O(n^{d/2})$ in the worst case. Chan and Ioannidis[5] proposed a new class of cube representations called Hierarchical Cubes. But the index mapping from a high-level "abstract" cube to a low-level "concrete" cube is too complicated for implementation. They did not verify the analytical results of their method experimentally. Another technique, called *Double Relative Prefix Sum*(DRPS)[10], was proposed to further reduce the update cost. By using three data structures, DRPS achieves the query cost of O($n^{1/3}$) per query and the update cost of O($n^{d/3}$) per update. Geffner et el.[9] proposed the *Dynamic Data Cube* which was designed by decomposing the prefix sum cube recursively. If the data cube is of high dimensions and high capacity, it is difficult to apply their approach since the approach incurs a high computation overhead. More recently, [8] proposed an efficient algorithm to make a drastic cut in the update propagation by using PC and an index structure called Δ-tree. Δ-tree is used to store the updated data, and PC is updated in batch when the Δ-tree becomes too large. While in a dynamic OLAP environment, cells of the data cube are frequently changed, i.e. hundreds thousand or millions of data items changed per day. This makes the Δ-tree too large, and results in poor range query performance, since the more frequently modified the data cell is, the more attention will be focused on it. In addition, the prefix sum cube will be bulk-updated with very high frequency. Fig.3 shows how the bulk-updated cost is affected when the size of data cube varies.

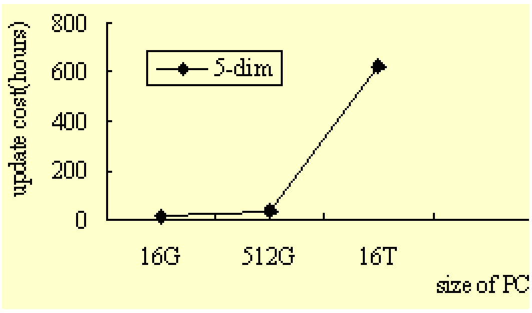


Fig. 3. Update cost for different sizes of PCs

For a given data cube, assume that each range query takes time t_q and each update takes time t_u. Both t_u and t_q are the time in the worst case. We further assume that the range sum query and the update operations on the data cube are mutually exclusive. [4] takes $t_q \times t_u$ as an optimization objective. This cost model is very useful when dealing with a data cube that has lot of queries but few updates. Clearly, this approach is inappropriate for dealing with the situation where queries and updates arise equally likely or periodically. [10] proposes another cost model. For a given time window, assume that the average numbers of range sum queries and updates to a data cube are known in advance. Thus, if there are n_q queries and n_u updates during the given time window, then, the total time used for both range queries and updates is $n_q t_q \times n_u t_u$.

3 Hierarchical Data Cube

In this section, we present a new cube storage structure, called hierarchical data cube, which can efficiently support range queries and updates. Like PS method [3], HDC technique also makes use of the *inverse property* of some aggregation operators. Accordingly range sum queries can be answered by adding and subtracting sums for appropriate regions which are anchored at (0, 0, …, 0). PS technique promises that any range sum query can be computed by combining up 2^d sums for ranges anchored at (0, 0, …, 0). We refer to a query as *prefix range query* if the query is anchored at (0, 0,…, 0). Since any arbitrary range query can be answered by adding and subtracting sums returned by a constant number of prefix range queries, thus the problem of computing the sum for an arbitrary range is reduced to the problem of efficiently computing the sum for any prefix query. We will therefore only describe how HDC solves this problem. In the following, we first introduce the main idea about constructing the hierarchical data cube, and then present algorithms for update and range sum query operations on HDC.

3.1 HDC and Its Construction

3.1.1 Hierarchical Partition Point

Definition 1. (Hierarchical Partition point) Let $A=\bigtimes_{i=1}^{d}[0, n\text{-}1]$ be a d-dimensional array with size of n^d and l be a non-negative integer. Then we call $P_i^l=\{(j*n/2^l)\text{-}1 \mid 1\le j<2^l, 2^l\le n, \text{ and } j \bmod 2\ne 0\}$ the l^{th} level set of *partition points* on dimension i.

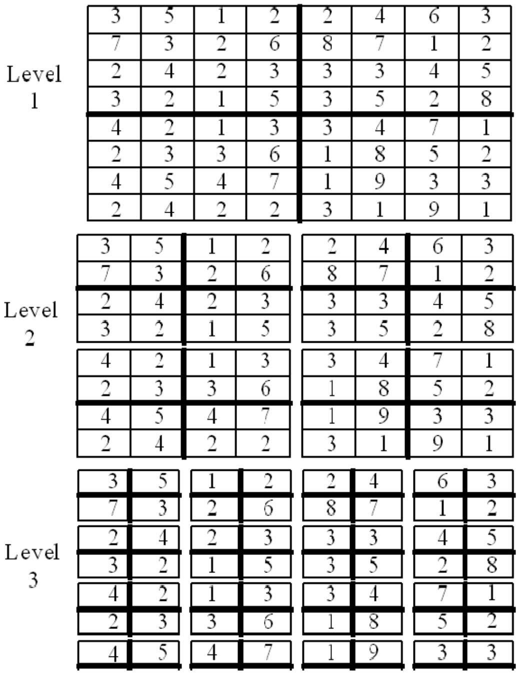


Fig. 4. Hierarchical partition points in different levels

$P_i^0=\{\text{n-1}\}$ is called the 0th level set of *partition points* on dimension i. It is obvious that for each dimension i, there are $logn+1$ sets of partition points: $P_i^0, P_i^1, \ldots, P_i^{\log n}$.

$P_i^0 \cup P_i^1 \cup \ldots \cup P_i^{\log n}$ consists of the domain of the *i*th dimension of *A*. that is $|P_i^0 \cup P_i^1 \cup \ldots \cup P_i^{\log n}| = n$.

Fig.4 depicts the hierarchical partition points for a 2-dimensional array *A* with size of 8×8. $P_1^0 = P_2^0 = \{7\}$ is the 0th level set of partition points on each dimension; $P_1^1 = P_2^1 = \{3\}$ is the 1th level set of partition points respectively; $P_1^2 = P_2^2 = \{1,5\}$ is the 2th level set of partition points respectively; and $P_1^3 = P_2^3 = \{0,2,4,6\}$ is the 3th level set of partition points respectively.

3.1.2 HDC and Its Construction

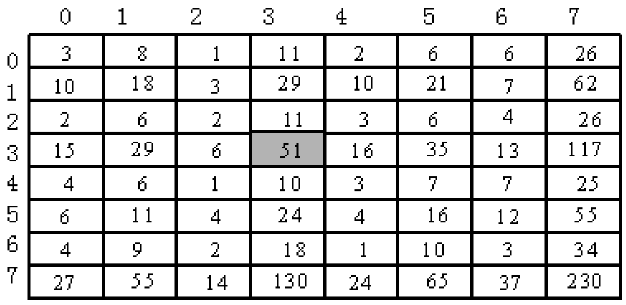
Definition 2. (Hierarchical Data Cube) The HDC of *A* is an *d*-dimensional array which has the same size as *A*. Given a cell $c(x_1,x_2,\ldots,x_d)$ in HDC, the value stored in *c* is determined as follows:

For each x_i, $0 \le i \le d$, there must exist a *k* ($0 \le k \le \log n$), satisfying $x_i \in P_i^k$. Then search p_i in $P_i^0 \cup P_i^1 \cup \ldots \cup P_i^{k-1}$, such that :

(1). $p_i < x_i$,and

(2). $\forall p_i' \in P_i^0 \cup P_i^1 \cup \ldots \cup P_i^{k-1}$, we have $p_i' \le p_i$.

Thus, *c* stores the sum of all cells in the region $[y_1,x_1] \times [y_2,x_2] \times \ldots \times [y_d,x_d] \subseteq A$(if p_i not found, $y_i=0$, else $y_i = p_i + 1$).



	0	1	2	3	4	5	6	7
0	3	8	1	11	2	6	6	26
1	10	18	3	29	10	21	7	62
2	2	6	2	11	3	6	4	26
3	15	29	6	51	16	35	13	117
4	4	6	1	10	3	7	7	25
5	6	11	4	24	4	16	12	55
6	4	9	2	18	1	10	3	34
7	27	55	14	130	24	65	37	230

Fig. 5. The HDC corresponding to the 2-dimensional array A in Fig. 4

Fig.5 shows the HDC corresponding to the 2-dimensional array *A* in Fig. 4. From the figure, we can see that the cell(3,3) stores the value of Sum(A[0,3]×[0,3]), since $x_1=3 \in P_1^1$, and there is no p_1 in P_1^0, satisfying $p_1<3$, so $p_1=0$, and the same with $p_2=0$. For the cell (4,4), since $x_1=4 \in P_1^3$, and there exists $p_1=3 \in P_1^0 \cup P_1^1 \cup P_1^2$, satisfying that $\forall p_1' \in P_1^0 \cup P_1^1 \cup P_1^2$, $p_1' \le p_1$ is true. Then $y_1=p_1+1=4$; Also $x_2=4 \in P_2^3$, and there exists $p_2=3 \in P_2^0 \cup P_2^1 \cup P_2^2$, satisfying that $\forall p_2' \in P_2^0 \cup P_2^1 \cup P_2^2$, $p_2' \le p_2$ is true. Then $y_2=p_2+1=4$. Thus, the cell(4,4) records the sum aggregate value of region A[4,4]×[4,4]. As the same, cell (4,5) records the sum aggregate value of region A[4,4]×[4,5], cell (4,6) records the sum aggregate value of region A[4,4]×[6,6], cell (4,7) records the sum aggregate value of region A[4,4]×[0,7], cell (5,4) records the sum aggregate value of region A[4,5]×[4,4], cell (5,5) records the sum aggregate value of region A[4,5]×[4,5], cell (5,6) records the sum aggregate value of region A[4,5]×[6,6], and cell (5,7) records the sum aggregate value of region A[4,5]×[0,7], and so on.

3.2 Range Queries and Updates on HDC

3.2.1 Range Queries

HDC can be used to answer any range sum query whose *target region* begins and ends at arbitrary cells in *A*. From [3], we know that any range sum query can be computed by combining up 2^d prefix queries. Thus the problem of computing the sum for an arbitrary range is reduced to the problem of efficiently computing the sum for any prefix query. The size of the query region will have no effects on the performance of range sum computation. The range sum query algorithm consists of two steps. First, it decomposes the *target regio*n *Q* into a set of sub-regions $\{q_1,q_2,\ldots,q_t\}$ such that each sum(q_j) can be obtained from a cell in HDC. Then in the second step, the algorithm accesses all these corresponding cells and calculates $\sum_{i=1}^{t} sum(q_i)$.

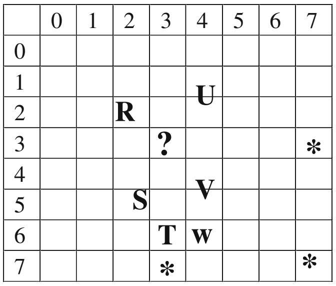


Fig. 6. Calculates the range sum query of region [0,6]×[0,4]

Suppose that we want to calculate the sum of the cells in the target region *Q* that begins at *A*(0,0) and ends at cell *A*(6,4). From Fig. 6, it's obvious that Q may be partitioned into 6 sub-regions: *R*, *S*, *T*, *U*, *V* and *W*. The aggregate sum value of the cells in Q can be obtained by Sum(*R*)+ Sum(*S*) + Sum(*T*) +Sum(*U*)+Sum(*V*)+Sum(*W*), where, Sum(*R*) is recorded in HDC(3,3), Sum(*S*) is recorded in HDC(5,3), Sum(*T*) is recorded in HDC(6,3), Sum(*U*) is recorded in HDC(3,4), Sum(*V*) is recorded in HDC(5,4), Sum(*W*) is recorded in HDC(6,4). Thus, Sum(*Q*)= HDC(3,3)+ HDC(5,3)+ HDC(6,3)+ HDC(3,4)+ HDC(5,4)+ HDC(6,4)=114.

From the example above, we can see that the key of answering range sum queries using HDC is how to decompose Q into these 6 sub-regions. First, on each dimension, the algorithm finds the maximal partition points, which falls in [0,6] and [0,4], in the *1th* partition point sets P_1^1 and P_2^1 respectively. "3" is obtained both from P_1^1 and P_2^1. Then, the algorithm cuts the interval [0,6] to two non-overlap sub intervals [0,3] and [4,6], and [0,4] to two non-overlap sub intervals [0,3] and [4,4], and then continues to find the maximal partition points, which falls in [4,6] and [4,4], in the *2th* partition point sets P_1^2 and P_2^2 respectively. "5" is obtained from P_1^2 and no point from P_2^2. Finally, the algorithm tries to find the maximal partition points, which falls in [6,6] and [4,4], in the *3th* partition point sets P_1^3 and P_2^3. “6” is obtained from P_1^3 and “4” is obtained from P_2^3. As the result, we get two partition point sets $P_1'=\{3,5,6\}$ and $P_2'=\{3,4\}$. P_1' partitions [0,6] into three intervals [0,3], [4,5] and [6,6]. P_2' partitions [0,4] into two intervals [0,3] and [4,4]. All these five intervals compose 6 sub-regions: [0,3]× [0,3],[0,3]×[4,4],[4,5]×[0,3],[4,5]×[4,4], [6,6]×[0,3] and [6,6]× [4,4], that is *R, S, T, U, V* and *W*.

Now we discuss the range sum query algorithm in detail. Suppose that the target region is $Q[0,h_1]\times[0,h_2]\times\ldots\times[0,h_d]$, the range sum query HDC_range_query(Q) consists of two steps:

Step one: decomposes Q into a set of sub-regions $Q_1, Q_2, \ldots, Q_t$, satisfying

(1). $Q= Q_1\cup Q_2\cup \ldots\cup Q_t$, and if $i\neq j$, then $Q_i\cap Q_j =\phi$;

(2). $\forall Q_i$, there must exist a cell HDC($x_1, x_2,\ldots, x_d$), such that sum(Q_i)=HDC($x_1, x_2,\ldots, x_d$).

Step two: for each Q_i, $1\le i \le t$, calculates Sum(Q_i) using HDC.

HDC_range_query(Q) can be formally described as follows:

```
HDC_range_query(Q)
Input: Prefix range query $Q[0,h_1]\times[0,h_2]\times\ldots\times[0,h_d]$,
Output: Sum($Q$).
Method:
(1) $SUM$=0;
  /* step one: decompose $Q$ into sub-regions $Q_1, Q_2, \ldots, Q_t$ */
  /* first, determine partition points $T_1,T_2,\ldots,T_d$ for Q on each dimension */
(2) FOR $i$=1 TO $d$ DO
(3)     $l_i$=0; $end$=false; $j$=0; $T_i=\phi$;
(4)     WHILE ($end$=false) DO
(5)     /* in $P_i^j$, find the maximal partition point $p$
          contained in [$l_i$,$h_i$], p=-1, if not found. */
(6)         $p$=max_partition_point($P_i^j$, $l_i$, $h_i$);
(7)         IF ($p\neq$-1) THEN
(8)             $T_i= T_i \cup\{p\}$; /* $T_i$ is an ordered set */
(9)             IF ($p\neq h_i$) THEN $l_i=p+1$;
(10)                    ELSE $end$=true;
(11)            $j=j+1$; /* END WHILE
  /* Then, obtain $t_i$ intervals :[0, $p_{i1}$],[$p_{i1}$+1, $p_{i2}$],…,[$p_{it_i-1}$+1, $h_i$] according to $T_i=\{p_{i1},p_{i2},\ldots,p_{it_i}\}$ ($1\le i\le d$, $p_{it_i}= h_i$) */
(12)        $l_1$=0; $l_2$=0;…; $l_d$=0;
(13)        FOR $i$=1 TO $d$ DO
(14)            $U_i =\phi$;
(15)            FOR $j$=1 TO $t_i$ DO
(16)                $U_i = U_i \cup\{$interval $u_j$:[$l_i$, $p_{ij}$] $\}$;
```

```
(17)            $l_i = p_{ij} + 1$;
(18)        ENDFOR;
  /* finally, surround sub-regoins $Q_1$, $Q_2$, …, $Q_t$ using
intervals in $U_1$, $U_2$,…, $U_d$ . */
(19)        $SQ$=NULL;
(20)        FOR  $\forall(1 \le j_1 \le t_1,\ 1 \le j_2 \le t_2, \dots,\ 1 \le j_d \le t_d)$  DO
(21)            $SQ = SQ \cup \{u_{1j_1} \times u_{2j_2} \times \dots \times u_{dj_d}\}$;
  /* Step two: calculate Sum($Q$) using $HDC$  */
(22)      FOR each $Q_r[l_{r1}, h_{r1}] \times [l_{r2}, h_{r2}] \times \dots \times [l_{rd}, h_{rd}]$
   $\in SQ$  DO
(23)          Sum($Q_r$)= HDC($h_{r1}$, $h_{r2}$ ,…, $h_{rd}$)
(24)          $SUM$= $SUM$+ Sum($Q_r$); /* calculate Sum($Q$) */
(25)        ENDFOR.
```

The key of the first step is decomposing Q into a set of sub-regions. The decomposing procedure is: for dimension i, beginning with the 0th level partition point set (in the example before, we omit to search the 0th level partition point set, since there is no point in P_1^0 and P_2^0 which falls in intervals [0,6] and [0,4]), try to search the maximal partition point $p \in P_i^0$, which falls in $[0,h_i]$. If found, then continue to search the maximal partition point p in the next level partition point set, which falls in $[p, h_i]$, otherwise search the maximal partition point p in the next level partition point set, which falls in $[0, h_i]$, ……., The processing stops when $p=h_i$. At last, we get d ordering sets $T_1, T_2, \dots, T_d$, where $T_i = \{p_{i1}, p_{i2}, \dots, p_{it_i}\}$. $\forall r,s$, if $r<s$, then $p_{ir} < p_{is}$. The points in $T_i (1 \le i \le d)$ partition $[0:h_i]$ into t_i intervals $[0: p_{i1}], [p_{i1}+1: p_{i2}], \dots, [p_{it_i-1}: p_{it_i}]$. Thus, all intervals on the d dimensions compose $t_1 \times t_2 \times \dots \times t_d$ non-overlap sub-regions that enclose Q.

3.2.2 Updates

The value of a cell $c(x_1, x_2, \dots, x_d)$ in A can be easily updated by searching HDC in bottom-up manner. The update algorithm first searches c, calculates the old value, determines the difference between the old and new values of the cell, and stores the new sum value into the cell. The difference value is used to update other related cells of HDC. For any cell $c'(y_1, y_2, \dots, y_d)$ in HDC, c' will be updated, if c' records the sum value of region $r \subseteq A$, and c is contained in r. Other cells that do not satisfy the condition are unaffected.

For example, the value of the cell (3,3) in Figure 4 is to be updated. From Fig. 6, we can see that the update algorithm need not only update the value of HDC(3,3), but also update all other cells denoted by ”*”: HDC(3,7), HDC(7,3) and HDC(7,7). The reason is that all the regions that these cells describe cover HDC(3,3), i.e. HDC(3,7) records the sum value of region A[0,3]×[0,7], and A[0,3]×[0,7] covers A(3,3). The key issue of the update algorithm is how to determine those cells related to the updated one. From the figures of 4 and 5, we know that each of the two dimensions has 4 partition levels: 0th, 1th, 2th and 3th partition levels. “3”$\in P_1^1$, also “3”$\in P_2^1$. First, beginning with the 0th partition level, the update algorithm searches all the partition points in P_1^0 and P_2^0, which fall in [3,7] respectively. “7” is obtained on each dimension. Then, the algorithm continues to search all the partition points in P_1^1 and P_2^1, which fall in [3,6] respectively. “3” is obtained on each dimension. Thus, we get a set of partition points on each dimension: T_1={7, 3},T_2={7, 3}. Those cells that coordinate at the points of $T_1 \times T_2$ are the cells that need to be modified in HDC.

The details of the update algorithm is given below:

```
    HDC_update(Q)
    Input: the updated cell c ($x_1$, $x_2$, … ,$x_d$) and its new value;
    Output: updated HDC.
      Method:
    (1)      Calculate the difference $\Delta c$.
    (2)      FOR  i=1  TO  d  DO
    (4)    $h_i$=n-1;  end=false;   j=0;   $T_i=\phi$;
    (5)    WHILE  (end=false)  DO
    (6)          /* in $P_i^j$, search partition point p contained in [$x_i$: $h_i$], if not found,  p=-1*/
    (7)        p=partition_point($P_i^j$, $x_i$, $h_i$);
    (8)        IF (p≠-1) THEN
    (9)           $T_i= T_i \cup \{p\}$;
    (10)          IF ($x_i \neq p$)   THEN  $h_i$=p-1;
    (11)          ELSE  end=true;
    (12)      j=j+1; /* END WHILE
    (13)  END FOR
    (14) FOR  $\forall(y_1 \in T_1,\ y_2 \in T_2,\ …,\ y_d \in T_d)$ DO
    (15)     HDC($y_1$, $y_2$ ,…, $y_d$)= $\Delta c$ +HDC($y_1$, $y_2$ ,…, $y_d$);
    (16) ENDFOR.
```

4 Complexity and Experimental Results

4.1 Complexity

In this section, we analyze the complexity of the HDC_range_query and HDC_update, including space complexity, and I/O complexity. Given a d-dimensional cube C with size of n^d, it’s obvious that the HDC of C has the same size of n^d, according to the definition of HDC. Thus, the space complexity of HDC is n^d.

Now we analyze the I/O complexity of HDC_range_query. HDC_range_query decomposes Q into a set of sub-regions in the first step which needs not I/O processing. In the second step, the algorithm executes d iteration during which log$n$ times of I/O is needed. Thus, in the worst case, the I/O complexity of HDC_range_query is $\mathbf{O}(\log^d n)$.

The analysis of HDC_update is similar with HDC_range_query. In the worst case, the I/O complexity of HDC_update is also $\mathbf{O}(\log^d n)$.

Table 1. Performance Comparison

Meth.	Query cost	Update cost	$cq \times cu$ model	$cq \times nq + cu \times nu$ model	Space requirement
Naive	$O(n^d)$	$O(1)$	$O(n^d)$		n^d
PS	$O(1)$	$O(n^d)$	$O(n^d)$	$O(K \times n^d)$	$2n^d$
RPS	$O(1)$	$O(n^{d/2})$	$O(n^{d/2})$	$O(K \times n^{d/2})$	$2 \cdot n^d + n^d \left(1 - \left(\frac{\sqrt{n}-1}{\sqrt{n}}\right)^d\right)$
DRPS	$O(n^{1/3})$	$O(n^{d/3})$	$O(n^{(1+d)/3})$	$O(K \times n^{d/3})$	$2 \cdot n^d + d \cdot n^{d-1/3}$
BRPS	$O(n^{1/3})$	$O(n^{d/3})$	$O(n^{(1+d)/3})$	$O(K \times n^{d/3})$	$2 \cdot n^d$
DDC	$O((2^d-1) \times \log^d n)$	$O(\log^d n)$	$O((2^d-1) \times \log^{2d} n)$	$O(2^d \times \log^d n)$	$\sum_{h=1}^{\log n} \left(\frac{n}{2^i}\right)^d \cdot \left(\left(\frac{n}{2^h}\right)^d - \left(\frac{n}{2^h}-1\right)^d\right)$ $>2n^d$
HDC	$O(\log^d n)$	$O(\log^d n)$	$O(\log^{2d} n)$	$O(K \times \log^d n)$	$2n^d$

Table 1 shows the comparisons of our proposed HDC technique with previous techniques with respect to cost of range queries, updates and the total cost under the two cost models.

PS is the first technique that has the minimal range query cost. But it's not applicable to the dynamic data warehouses due to its high update cost and high total cost under the two cost models. Thus, several techniques have been proposed to improve it by adding additional storage space, such as RPS, DRPS, BRPS and DDC etc. From table 1, we can see that our HDC method outperforms other known approaches under its range query cost, update cost and both cost models, and it needs the same additional storage space with PS.

4.2 Experimental Results

In table 1, DDC is the best one with minimal total cost among the previous existed techniques. So we just compare HDC with DDC in range query cost and total cost during the following experiments. In the following experiments, we use the cells affected by range query and update operations to denote the I/O cost. There are two factors that affect the performance of range queries and updates. The first one is the size of multi-dimensional cube; the second one is the dimensionality, the number of dimensions in a cube. We compare HDC with DDC in the two factors. In the experiments, the benchmark datasets were synthetic, being randomly generated.

4.2.1 Range Query Comparison

First, we vary the size of multi-dimensional cube while fixing the number of dimensions. We adjust the size of cube by varying the cardinality of each dimension. In this experiment, we generate three groups of dataset with *d*=3,4 and 5. Without loss of generality and for convenient, we assume that each dimension in each dataset has the same cardinality, which is the necessary requirement of DDC, but not true for HDC.

Fig. 7, 8 and 9 show the experimental results for different number of dimensions, where the horizontal ordinate indicates the cardinality of the dimension ranging from 64 to 512, the vertical ordinate indicates the number of cells visited by the range queries. We can see from the figures that when the number of dimensions is small, i.e. 3, the varying of the cube size has little effect on range query performance. While, for the datasets with higher dimensionality, the number of the cells, visited by DDC, will increase rapidly with the enlargment of cube size. See the curve in Fig. 9, when *d*=5.

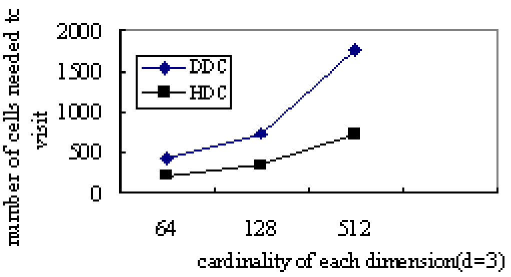


Fig. 7. Varying the size of datasets when d=3

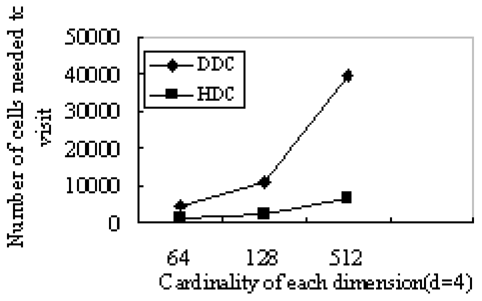


Fig. 8. Varying the size of datasets when d=4

The second group of experiments is to test how the dimensionality affects range query performance. We do this on three kinds of datasets with different amount of data. We call the dataset large-scaled dataset if it has over 10^{12} data cells. We call the dataset medial-scaled dataset if it has over 10^{10}~10^{11} data cells. We call the dataset small-scaled dataset if it has less than 10^9 data cells.

In this experiments, the amounts of the three kinds of datasets are 2^{40}, 2^{36} and 2^{32} respectively. The experimental results are depicted in Fig. 10, 11, and 12. The figures show that varying the dimensionality can only incurs the range query performance changing slowly on the same-scaled datasets using HDC. But it's quite different for

DDC. When the scale is varied from small to large, the performance of range queries are influenced greatly. This can be seen from Fig. 12.

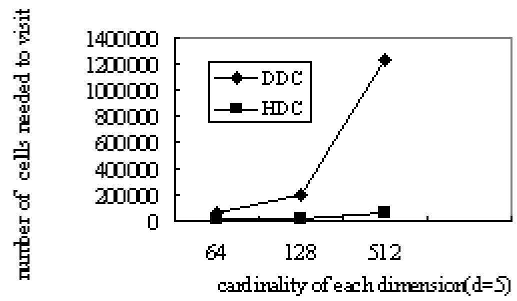


Fig. 9. Varying the size of datasets when d=5

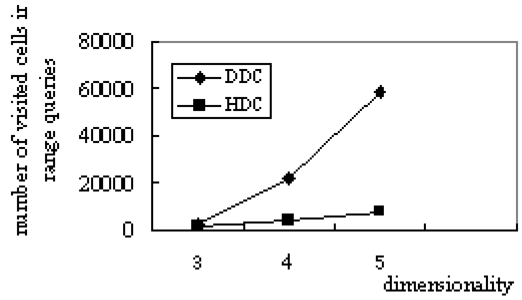


Fig. 10. Varying dimensionality on small-scaled datasets

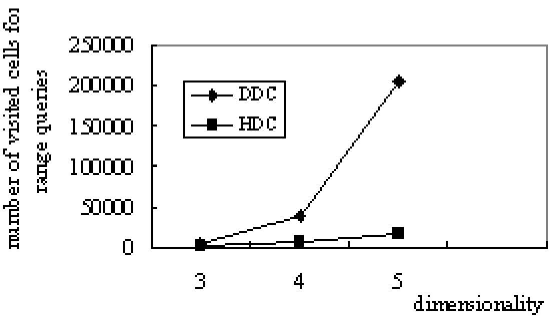


Fig. 11. Varying dimensionality on medial-scaled datasets

4.2.2 Total Cost under Two Cost Models

We give the experiments on the datasets being same with the first experiment in 4.2.1, comparing the total performance under the two cost models. Fig. 13 and 14 show the ratio of (number of visited cells using DDC)/(number of visited cells using HDC). Fig.13 indicates that under the *cq*×*cu* model, there is little difference between the performance of HDC and DDC, when the dataset scale is small. But the difference becomes great as the dimensionality increased (i.e. d=5). The performance of HDC is quite superior to DDC.

Fig. 14 shows that under the *cq*×*nq* + *cq*×*nu* model (assume *nq*=*nu*=10), there is little difference between HDC and DDC, when the dimensionality is small, i.e. d=3 or 4. But HDC outperforms DDC obviously, when both the dimensionality and the cardinality increase.

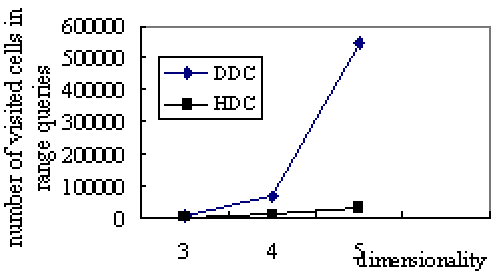


Fig. 12. Varying dimensionality on large-scaled datasets

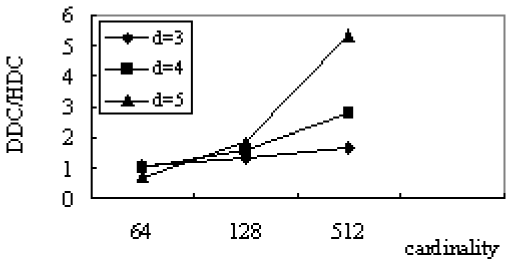


Fig. 13. Comparison under $C_q{\times}C_u$

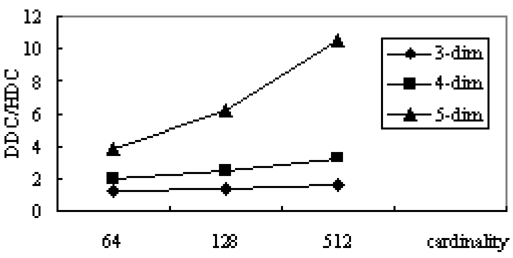


Fig. 14. Comparison under $N_q{\times}C_u+N_q{\times}C_u$

5 Conclusions

Due to an increasing demand for OLAP and data cube applications, efficient calculation of range queries has become more important in recent years. Several techniques have been developed to answer the range sum queries efficiently, however these methods still may incur a high update cost. In this paper, we present the Hierarchical Data Cube, a new method for handling range sum queries in data cubes which has the time complexity $O(\log^d n)$ for both range queries and updates. Under the two cost models, our approach achieves the best overall complexity of update cost and query cost.

Still, there exist some issues needed to be considered in the future, such as how to select the number of partitions in a hierarchical data cube, and when the number of partitions is set up, how to selec the "optimal" number of partitions. In addition, since parallelization is a very important technique for fast query processing, especially for OLAP queries dealing with massive data. So in future work, we also investigate parallel techniques to process OLAP queries, and develop parallel indexing mechanisms for data cubes.

References

1. Codd E. F.: Providing OLAP(on-line analytical processing) to user-analysts: An IT mandate, Technical report, E. F. Codd and Associates, 1993.
2. Gray J., Bosworth A., Layman A., Pirahesh H.: Data cube: A relational aggregation operator generating group-by, cross-tab and sub-total. In *Proc. Of the 12th ICDE Conf.* 1996. 152–159
3. Ho C.T., Agrawal R., Megiddo R., and Srikant R.: Range Queries in OLAP Data Cubes. In *Proceedings of the Intl. ACM SIGMOD Confference*, 1997, 73–88.
4. Geffner S., Agrawal D., Abbadi A., and Smith T.: Relative Prefix Sums: An Efficient Approach for Querying Dynamic OLAP Data Cubes. In *Proceedings of the 15th Intl. Conference on Data Engineering*, 1999, 328–335.
5. Chan C.Y. and Ioannidis Y.E.: Hierarchical Cubes for Range-Sum Queries. In *Proceedings of the 25th VLDB Conference*, 1999, 675–686.
6. Lee S. Y., Ling T. W., Li H. G.: Hierarchical compact cube for range-max queries. In *Proc. Of the 26th VLDB Conf.* 2000, 232–241.
7. Li H. G., Ling T. W., Lee S. Y., Loh Z. X.: Range sum queries in dynamic OLAP data cubes. 3th International Symposium on Cooperative Database Systems for Advanced Applications(CODAS'01), 2001, 74–81.
8. Chun S. J., Chung C. W., Lee J. H., Lee S. L.: Dynamic update cube for range-sum queries In Proc. of 27th VLDB Conf. 2001, 521–530.
9. Geffner S., Agrawal D., El Abbadi A., The Dynamic Data Cube, In Proc. of Int'l Conference on Extending Database Technology, 2000, LNCS 1777, 237–253.
10. Liang W., Wang H., Orlowska M. E.. Range queries in dynamic OLAP data cubes. *Data and Knowledge Engineering 34*, 2000, 21–38.

Evaluation of Common Counting Method for Concurrent Data Mining Queries[*]

Marek Wojciechowski and Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
{marek,mzakrz}@cs.put.poznan.pl

Abstract. Data mining queries are often submitted concurrently to the data mining system. The data mining system should take advantage of overlapping of the mined datasets. In this paper we focus on frequent itemset mining and we discuss and experimentally evaluate the implementation of the Common Counting method on top of the Apriori algorithm. The general idea of Common Counting is to reduce the number of times the common parts of the source datasets are scanned during the processing of the set of frequent pattern queries.

1 Introduction

Data mining, also referred to as database mining or knowledge discovery in databases (KDD), aims at discovery of useful patterns from large databases or warehouses. Nowadays we are witnessing the evolution of data mining environments from specialized tools to multi-purpose data mining systems offering some level of integration with existing database management systems. Data mining can be seen as advanced querying, where a user specifies the source dataset and the requested pattern constraints, then the system chooses the appropriate data mining algorithm and returns the discovered patterns to the user. One of the most serious problems concerning data mining queries is long response time. Current systems often consume minutes or hours to answer single queries.

In practical applications, data mining queries are often executed during nights, when system activity is low. Sets of queries are scheduled and then automatically evaluated by a data mining system. It is possible that the data mining queries delivered to the system are somehow similar, e.g., their source datasets overlap. Such queries can be executed serially, however, the fact that the datasets overlap may help in parallelizing the query execution.

In [13] we have proposed various methods for batch processing of data mining queries for frequent itemset discovery. In order to improve the overall performance of the batched data mining queries, the methods tried to employ the fact that their source datasets overlapped. One of the methods, called *Mine Merge*, partitioned the set of

[*] This work was partially supported by the grant no. 4T11C01923 from the State Committee for Scientific Research (KBN), Poland.

batched queries into the set of simple isolated queries. Another method, called *Common Counting*, integrated the phases of support counting for candidate itemsets. Both methods resulted in significant database I/O reduction.

In this paper we present our experiences in implementing and evaluating the *Common Counting* method for concurrent data mining queries. We discuss the influence of various parameters on the efficiency of *Common Counting*. Experimental results are given to prove the advantages of the *Common Counting* method in the context of concurrent data mining queries.

1.1 Related Work

The problem of mining frequent itemsets and association rules was introduced in [1] and an algorithm called *AIS* was proposed. In [3], two new algorithms were presented, called *Apriori* and *AprioriTid*, that achieved significant improvements over *AIS* and became the core of many new algorithms for mining association rules. *Apriori* and its variants first generate all frequent itemsets (sets of items appearing together in a number of database records meeting the user-specified support threshold) and then use them to generate rules. *Apriori* and its variants rely on the property that an itemset can only be frequent if all of its subsets are frequent.

Since it has been observed that generation of association rules from frequent itemsets is a straightforward task, further research focused on alternative methods for frequent itemset mining. In [14], a different algorithm called *Eclat* has been proposed. *Eclat* generates lists of itemset identifiers to efficiently generate and count frequent itemsets. Recently, a new family of pattern discovery algorithms, called pattern-growth methods (see [7] for a review), has been developed for discovery of frequent itemsets (and other classes of patterns). The methods project databases based on the currently discovered frequent patterns and grow such patterns to longer ones in corresponding projected databases. Pattern-growth methods are supposed to perform better than *Apriori*-like algorithms in case of low minimum support thresholds. Nevertheless, practical studies [10] show that for real datasets *Apriori* and *Eclat* might still be a more efficient solution.

The notion of data mining queries (or *KDD* queries) was introduced in [8]. The need for Knowledge and Data Management Systems (KDDMS) as second generation data mining tools was expressed. The ideas of application programming interfaces and data mining query optimizers were also mentioned. Several data mining query languages that are extensions of *SQL* were proposed [4][6][9][11][12].

1.2 Basic Definitions

Frequent Itemsets. Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. Let a non-empty set of items T be called an *itemset*. Let D be a set of variable length itemsets, where each itemset $T \subseteq L$. We say that an itemset T *supports* an item $x \in L$ if x is in T. We say that an itemset T *supports* an itemset $X \subseteq L$ if T supports every item in the set X. The *support* of the itemset X is the percentage of T in D that support X. The problem of mining frequent itemsets in D consists in discovering all itemsets whose support is above a user-defined support threshold.

Apriori Algorithm. *Apriori* is an example of a level-wise algorithm for frequent itemset discovery. It makes multiple passes over the input data to determine all frequent itemsets. Let L_k denote the set of frequent itemsets of size k and let C_k denote the set of candidate itemsets of size k. Before making the k-th pass, *Apriori* generates C_k using L_{k-1}. Its candidate generation process ensures that all subsets of size $k-1$ of C_k are all members of the set L_{k-1}. In the k-th pass, it then counts the support for all the itemsets in C_k. At the end of the pass all itemsets in C_k with a support greater than or equal to the minimum support form the set of frequent itemsets L_k. Figure 1 provides the pseudocode for the general level-wise algorithm, and its *Apriori* implementation. The *subset(t, k)* function gives all the subsets of size k in the set t.

This method of pruning the C_k set using L_{k-1} results in a much more efficient support counting phase for *Apriori* when compared to the earlier algorithms. In addition, the usage of a hash-tree data structure for storing the candidates provides a very efficient support-counting process.

```
$C_1$ = {all 1-itemsets from $D$}
for ($k$=1; $C_k \neq \emptyset$; $k$++)
        count($C_k$, $D$);
    $L_k = \{c \in C_k \mid c.count \geq minsup\}$;
    $C_{k+1}$ = generate_candidates($L_k$);
$Answer = \bigcup_k L_k$;
```

```
$L_1$ = {frequent 1-itemsets}
for ($k$ = 2; $L_{k-1} \neq \emptyset$; $k$++)
    $C_k$ = generate_candidates($L_{k-1}$);
    forall tuples $t \in D$
            $C_t = C_k \cap$ subset($t$, $k$);
            forall candidates $c \in C_t$
                    $c.count$++;
    $L_k = \{c \in C_k \mid c.count \geq minsup\}$
$Answer = \bigcup_k L_k$;
```

Fig. 1. A general level-wise algorithm for association discovery (left) and its Apriori implementation (right)

2 Preliminaries and Problem Statement

Data Mining Query. A data mining query is a tuple $DMQ = (R, a, \Sigma, \Phi)$, where R is a relation, a is an attribute of R, Σ is a condition involving the attributes of the relation R, Φ is a condition involving discovered patterns. The result of the data mining query is a set of patterns discovered in $\pi_a \sigma_\Sigma$ and satisfying Φ.

Example. Given the relation R_1 shown in Fig. 2, the result of the data mining query DMQ_1=(R_1, *"basket"*, *"id>5 AND id<10"*, *"minsup ≥ 3"*) is shown in Fig. 3.

Problem Statement. Given a set $S = \{DMQ_1, DMQ_2, \ldots, DMQ_n\}$ of data mining queries, where $DMQ_i = (R, a, \Sigma_i, \Phi_i)$ and $\forall_i \exists_{j \neq i}\, \sigma_{\Sigma i}(R) \cap \sigma_{\Sigma j}(R) \neq \emptyset$, the goal is to minimize the I/O cost and the CPU cost of executing S.

```
R1: id  basket
    --------
     1  a,b,c
     4  a,c
     6  d,f,g
     7  f,g,k,m
     8  e,f,g
    15  a,f
```

Fig. 2. Example relation R_1

```
result of DMQ1:

{f}
{g}
{f,g}
```

Fig. 3. DMQ_1 query result

2.1 Motivating Example

Consider a relation *Sales(uad, basket, time)* to store purchases made by users of an internet shop. For every visit to a shop, we store the user's internet address (*uad*), the products purchased (*basket*) and the time of the visit (*time*). Since datasets of this kind tend to be very large, there is a need for automated analysis of their contents. One of the data mining methods that proved to be useful for such analysis is associations discovery. Assume a shop manager is interested in finding sets of products that were frequently co-occurring in the users' purchases. The shop manager plans to create two reports: one showing the frequent sets that appeared in more than 350 purchases in January 2002 and one showing the frequent sets that appeared in more than 20 purchases made by customers from France. The two data mining queries shown below are required to construct the response.

DMQ_A=(Sales, "basket", "time between '01-01-02' and '01-31-02'", "minsup > 350")
DMQ_B=(Sales, "basket", "uad like '%.fr'", "minsup > 20")

If the size of the *Sales* relation is very large, each of the above data mining queries can take a significant amount of time to execute. Part of this time will be spent on reading the *Sales* relation from disk in order to count occurrences of candidate itemsets. Notice that the sets of blocks to be read by the two data mining queries may overlap. If we try to merge the processing of the two data mining queries, we can reduce redundancy resulting from this overlapping. In the remaining of this paper we will use this example to illustrate our method.

3 Cost Analysis of Level-Wise Algorithms

In order to analytically compare the complexity of the method considered, first we derive the execution cost functions for a generic level-wise algorithm. We make the following assumptions: (1) the size of the database is much larger than the size of all candidate itemsets, (2) the size of all candidate itemsets can be larger than the memory size, and (3) frequent itemsets fit in memory. The notation we use is given in Table 1.

Table 1. Notation used in cost models

M	main memory size (blocks)
$\|D\|$	number of itemsets in the database
$\|\|D\|\|$	size of the database (blocks)
$\|C_i\|$	number of candidate itemsets for step *i*
$\|\|C_i\|\|$	size of all candidate itemsets for step *i* (blocks), $\|\|C_i\|\|<<\|\|D\|\|$, $\|\|C_i\|\|<M$
$\|L_i\|$	number of frequent itemsets for step *i*, $\|L_i\|<\|C_i\|$
$\|\|L_i\|\|$	size of all frequent itemsets for step *i* (blocks), $\|\|L_i\|\|<M$

The cost of performing the general level-wise association discovery algorithm is as follows:

1. **Candidate Counting-Pruning.** Candidate itemsets must be read from disk in portions equal to the available memory size. For each portion, the database must be scanned to join itemsets from C_i with itemsets from *D*. Next, the candidate itemsets with support greater or equal to *minsup* become frequent itemsets and must be written to disk. The I/O cost of a single iteration *i* is the following:

$$\text{cost}_{I/O} = \|C_i\| + \frac{\|C_i\|}{M}\|D\| + \|L_i\|$$

   The dominant part of the CPU cost is join condition verification. For the simplicity, we assume the cost of comparing two itemsets does not depend on their sizes and equals 1. Thus, the CPU cost of a single iteration *i* is the following:

$$\text{cost}_{CPU} = |C_i||D|$$

2. **Candidate Generation.** Frequent itemsets from the previous iteration must be read from disk, joined in memory, and saved as new candidate itemsets. The I/O cost of a single iteration *i* is the following:

$$\text{cost}_{I/O} = \|L_i\| + \|C_{i+1}\|$$

   The CPU cost of this phase of the algorithm is the following:

$$\text{cost}_{CPU} = |L_i||L_i|$$

Therefore, if *K* is the number of iterations, the overall cost of the level-wise algorithm is as follows:

$$\text{cost}_{I/O} = \sum_{i=1}^{K}\left(\|C_i\| + \frac{\|C_i\|}{M}\|D\| + 2\|L_i\| + \|C_{i+1}\|\right)$$

$$\text{cost}_{CPU} = \sum_{i=1}^{K}\left(|C_{i+1}||D| + |L_i|^2\right)$$

4 Common Counting Method

When two or more different DMQs count their candidate itemsets in the same part of the database, only one scan of the common part of the database is required and during that scan candidates generated by all the queries referring to that part of the database are counted. For the sake of simplicity we formally present the *Common Counting* method in the context of *Apriori* algorithm for two concurrent DMQs only (Fig. 4). It is straightforward to extend the technique to support more than two DMQs.

```
$C_1^A$ = {all 1-itemsets from $D^A$}
$C_1^B$ = {all 1-itemsets from $D^B$}
for ($k$=1; $C_k^A \cup C_k^B \neq \emptyset$; $k$++)
    if $C_k^A \neq \emptyset$ count($C_k^A$, $D^A - D^B$);
    if $C_k^B \neq \emptyset$ count($C_k^B$, $D^B - D^A$);
    count($C_k^A \cup C_k^B$, $D^A \cap D^B$);
    $L_k^A = \{c \in C_k^A \mid c.count \geq minsup^A\}$;
    $L_k^B = \{c \in C_k^B \mid c.count \geq minsup^B\}$;
    $C_{k+1}^A$ = generate_candidates($L_k^A$);
    $C_{k+1}^B$ = generate_candidates($L_k^B$);
$Answer^A = \bigcup_k L_k^A$;
$Answer^B = \bigcup_k L_k^B$;
```

Fig. 4. Model of the Common Counting method

During the integrated counting, the candidates from all the DMQs are loaded into the main memory. Next, the database is scanned and for each itemset from the database the supported candidate counters for all the relevant queries are incremented. If the candidates' set does not fit into memory, it is loaded in parts and the database is scanned multiple times (for the combined candidate set we apply the scheme proposed in the context of the original *Apriori* algorithm in [3]). After the candidate supports are calculated, frequent itemsets are derived in a traditional way.

The *Common Counting* method is directly applicable to all level-wise algorithms (e.g., *Apriori*)[1]. The advantage of the method is database scan reduction because the common part of the database must be logically read only once.

Different ways of implementing the candidate counting step are possible. In this paper we propose the following solution. The database is logically divided into disjoint partitions such that each part is a subset of the source dataset for one or more DMQs. In the candidate verification phase the database is scanned sequentially. Before scanning each of the partitions, candidates for all the queries referring to that partition have to be available. If for a given query the candidates have not been generated while processing previous partitions, then they have to be generated. We do not combine the sets of candidates for the collection of queries - for each query a separate candidate hash-tree is built. If the candidates for a given partition do not fit into available memory, they are processed in parts and the database partition is scanned several

[1] It should be noted that Common Counting can also be applied to algorithms that use Apriori-like generate-and-test scheme in one of their phases (e.g., Eclat)

times (as in the original *Apriori*). After processing a database partition, the candidate hash-trees are swapped to disk only if the system is short of memory. Candidate hash-trees for queries that will not be needed for the highest number of subsequent partitions are first to be swapped.

We see two alternatives to the candidate counting scheme described above based on the idea of combining candidate sets. Firstly, the candidates for all the queries can be combined and stored in one hash-tree with each candidate having several counters (one for each query). Since such a structure will contain candidates that should be counted in some partitions but not considered in others, a list of relevant database partitions should be associated with each candidate itemset. Efficiency of the scheme will likely depend on the number of common candidates between the queries.

Secondly, we can modify the above scheme by combining the sets of candidates of the queries before processing each database partition, considering only the candidates of the relevant queries. This approach minimizes the memory usage but requires several costly operations of building and then destroying hash-trees. Evaluation of the two alternative candidate counting schemes is beyond the scope of this paper and is regarded as a topic for future research.

Example. Let us consider the following example of the *Common Counting* method. Using the database selection conditions from the Section 2.1, we construct three separate dataset definitions:

1.
```
select basket
from   sales
where  time between '01-01-02' and '01-31-02'
  and  NOT uad like '%.fr'
```

2.
```
select basket
from   sales
where  time between '01-01-02' and '01-31-02'
  and  uad like '%.fr'
```

3.
```
select basket
from   sales
where  NOT time between '01-01-02' and '01-31-02'
  and  uad like '%.fr'
```

Next, we scan the first query's result in order to count DMQ^A candidate itemsets, then we scan the second query's result in order to count both DMQ^A and DMQ^B candidate itemsets, finally we scan the third query's result in order to count DMQ^B candidate itemsets. Notice that none of the database blocks needed to be read twice, on the condition that candidate itemsets fit in memory.

Let us analyze the cost of the level-wise phase of the *Common Counting* method. Candidate itemsets of DMQ^A must be read, joined with D^A-D^B, counted, and saved to disk. Also, candidate itemsets of DMQ^B must be read, joined with D^B-D^A, counted, and saved to disk. Next, all candidates of DMQ^A and DMQ^B must be read, joined with $D^A \cap D^B$, counted, and saved to disk. The candidate itemsets with support greater or equal to, respectively, $minsup^A$ or $minsup^B$, become frequent itemsets and are written to disk.

In order to generate new candidate itemsets, all frequent itemsets must be read from disk and new candidate itemsets must be written to disk. Therefore, the I/O cost of this method is the following:

$$\mathrm{cost}_{I/O} = \sum_{i=1}^{\max(K^A,K^B)} \left(\begin{array}{l} 3\left\|C_i^A\right\| + \dfrac{\left\|C_i^A\right\|}{M}\left\|D^A - D^B\right\| + 3\left\|C_i^B\right\| + \dfrac{\left\|C_i^B\right\|}{M}\left\|D^B - D^A\right\| \\ + \dfrac{\left\|C_i^A\right\| + \left\|C_i^B\right\|}{M}\left\|D^A \cap D^B\right\| + 2\left\|L_i^A\right\| + 2\left\|L_i^B\right\| + \left\|C_{i+1}^A\right\| + \left\|C_{i+1}^B\right\| \end{array} \right)$$

Similarly, the CPU cost is as follows:

$$\mathrm{cost}_{CPU} = \sum_{i=1}^{K^A}\left(\left|C_{i+1}^A\right|\left|D^A - D^B\right| + \left|L_i^A\right|^2\right) + \sum_{i=1}^{K^B}\left(\left|C_{i+1}^B\right|\left|D^B - D^A\right| + \left|L_i^B\right|^2\right) +$$
$$\sum_{i=1}^{\max(K^A,K^B)}\left(\left(\left|C_{i+1}^A\right| + \left|C_{i+1}^B\right|\right)\left|D^B \cap D^A\right|\right)$$

5 Performance Analysis

In order to evaluate performance of the *Common Counting* method in the context of frequent itemset mining we performed several experiments on synthetic and real datasets. The synthetic datasets were generated by means of the *GEN* generator from the *Quest* project [2]. The real datasets that we have used come from the UCI KDD Archive [5]. Here we report results obtained on two real datasets from UCI KDD Archive: Movies[2] and MSWeb[3] (Microsoft Anonymous Web Data), and two synthetic datasets (denoted as Gen1 and Gen2). Table 2 presents basic characteristics of these datasets (for both synthetic datasets the remaining *GEN* parameters not listed in Table 2 were set to the following values: the number of patterns was set to 500 and the average pattern length to 3).

Table 2. Datasets used in experiments

	Gen1	**Gen2**	**Movies**	**MSWeb**
Number of sets in DB	100000	100000	8040	32710
Avg length of set in DB	5	8	5	3
Number of different items in DB	1000	10000	14561	285

We implemented *Common Counting* on top of the classic *Apriori* algorithm using the candidate counting scheme with a separate candidate hash-tree for each query. The experiments were conducted on a PC with AMD Duron 1200 MHz processor and 256 MB of main memory. The datasets used in all experiments resided in flat files on a

[2] http://kdd.ics.uci.edu/databases/movies/movies.html

[3] http://kdd.ics.uci.edu/databases/msweb/msweb.html

local disk (disk cache was disabled). The amount of available main memory in the reported experiments was big enough to store all the candidates generated by all the queries in a given iteration. (We discuss the impact of the amount of available memory at the end of this section.)

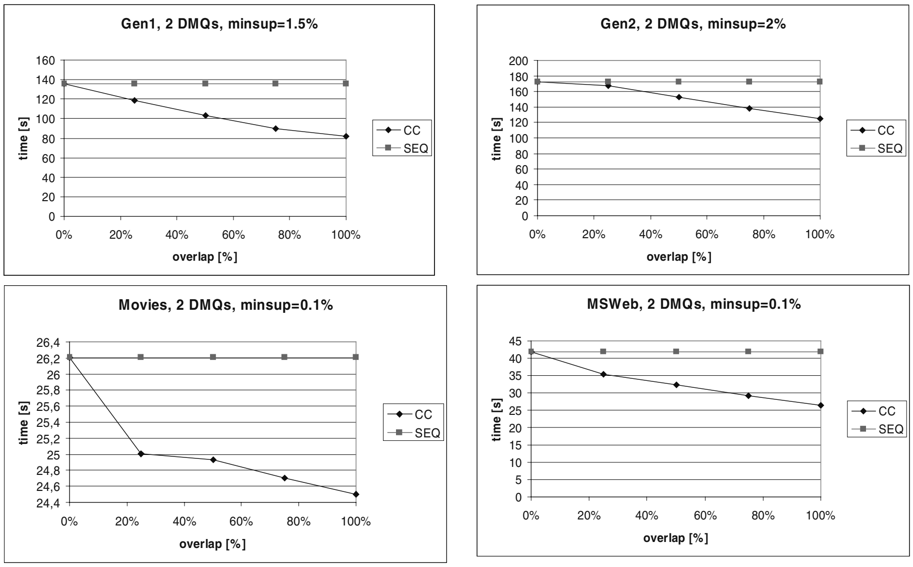


Fig. 5. Performance of Common Counting for various levels of dataset overlapping

In the first series of experiments we varied the level of overlapping of source datasets for the case of two queries. Both queries operated on datasets of the same size and containing the same data, the support threshold was also the same for both queries. We changed the level of overlapping from 0% to 100%. Overlapping is expressed as the ratio of the size of the common part of the database to the size of the source dataset of any of the two queries (recall that the experiments were conducted on pairs of identical queries). Figure 5 shows the processing times of *Common Counting* denoted as CC compared to sequential processing of the queries using *Apriori* denoted as SEQ. For all tested datasets *Common Counting* outperformed sequential execution of *Apriori* and the bigger the overlapping of source datasets for the queries the bigger the performance gap between the two methods. However, the performance gains thanks to *Common Counting* depend on the characteristics of the data and the minimum support threshold. *Common Counting* applied to the Movies dataset led to relatively small performance gains since in case of this dataset there was a huge number of candidates generated in the second iteration, and only a few of them turned out to be frequent leading to a small number of iterations (and database scans). Thus, the time spent on candidate verification dominated the time needed to scan the database which is optimized by *Common Counting*.

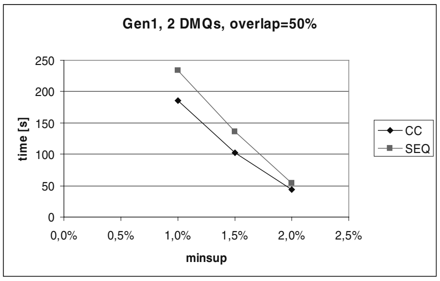


Fig. 6. Performance of Common Counting for different minimum support thresholds

In the next experiment we varied the minimum support threshold (same for both queries). Figure 6 presents the impact of the minimum support threshold for the Gen1 dataset, overlapping of 50%, and two queries. Interestingly, the relative performance of *Common Counting* was better for the threshold of 1.5% than for 1% or 2%. This means that neither increasing nor decreasing the support threshold will always work in favor of Common Counting. Such behavior is caused by the fact that the minimum support threshold has an impact on the number of frequent patterns (and indirectly on the number of processed candidates) and the number of iterations (related to the size of the largest frequent pattern). What is guaranteed is that increasing the minimum support threshold will not increase neither the number of candidates not the number of iterations. Similarly, decreasing the minimum support threshold will not decrease neither the number of candidates not the number of iterations. However, changing the support threshold can affect one of the values stronger than the other. Thus, increasing or decreasing the minimum support threshold can change the balance between the time needed to count the candidates (operation not optimized by *Common Counting*) and the time needed to read data for disk (operation optimized by *Common Counting*).

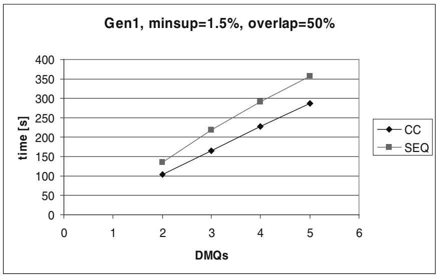


Fig. 7. Performance of Common Counting for different number of queries

The next goal of the experiments was testing the efficiency of *Common Counting* for different number of queries processed concurrently. Again, for the sake of simplicity, we performed the tests on collections of identical queries (the same dataset and support threshold) increasing the number of queries from 2 to 5. We considered only one overlapping configuration (from many possible for more than two queries), where there was one partition common to all the queries and the rest of partitions were analyzed each by exactly one query (in such configuration we could express overlapping in the same manner as in the case of two queries). Figure 7 presents the impact of the number of concurrently processed queries for the Gen1 dataset, overlapping of 50%, and minimum support threshold of 1.5%. Again *Common Counting* outperformed *Apriori* in all the cases.

All the results reported above were obtained on collections of identical queries (same data, same minimum support threshold). If the queries that are to be performed concurrently differ in the minimum support threshold and/or the source data (which is going to be the case in practical applications), then the performance gains thanks to *Common Counting* are smaller than in case of identical queries if the queries require different number of iterations to complete (performance gains are most significant if all scans of the common part of the database are exploited by all overlapping queries). It should be noted that the number of *Apriori* iterations for a given query is not known until the query completes, and depends on the characteristics of the data and the minimum support threshold.

As we mentioned earlier, in the experiments reported above all the candidates processed in a given iteration fit into main memory, which is realistic for today's computers (in case of reasonable support thresholds) but not required neither by *Apriori* nor by our *Common Counting* method. Recall that if the candidates to be verified in a given partition of database in a given iteration do not fit into main memory, then they have to be processed in parts and the corresponding partition is scanned more than once. Thus, limiting the amount of main memory can degrade performance of original *Apriori* as well as *Common Counting*. However, the relative performance of *Common Counting* compared to sequential *Apriori* may improve, degrade, or stay unchanged with the increase of available memory. The actual performance gains depend on the ratio of the amount of data read from disk by the *Common Counting* method to the amount of data read from disk by the queries executed sequentially.

6 Conclusions

We have presented our experiences in implementing and evaluating the *Common Counting* method for concurrent frequent itemset queries. The *Common Counting* method is specific to the class of algorithms that at least in some phases need to count the occurrences of candidates in the source dataset. The method exploits the fact that the source datasets of queries that are to be processed can overlap, and consists in reducing the number of scans of parts of the database that are common to two or more queries.

We have implemented the *Common Counting* method on top of the classic *Apriori* algorithm and experimentally tested its efficiency with respect to various parameters. Our experiments showed that the method is efficient and usually leads to significant performance gains compared to sequential processing of queries.

In the future we plan to further investigate the *Common Counting* scheme focusing on issues regarding possible strategies of managing candidate sets of queries processed concurrently. We also plan to work on methods of processing of sets of data mining queries that do not depend on a particular mining scheme (such as *Apriori* in case of *Common Counting*).

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conf. on Management of Data (1993)
2. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon (1996)
3. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
4. Ceri S., Meo R., Psaila G.: A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases (1996)
5. Hettich S., Bay S. D.: The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science (1999)
6. Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R.: DBMiner: A System for Mining Knowledge in Large Relational Databases. Proc. of the 2nd KDD Conference (1996)
7. Han J., Pei J.: Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. SIGKDD Explorations, December 2000 (2000)
8. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
9. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference (1996)
10. Zheng Z., Kohavi R., Mason L.: Real World Performance of Association Rule Algorithms. Proc. of the 7th KDD Conference (2001)
11. Morzy T., Wojciechowski M., Zakrzewicz M.: Data Mining Support in Database Management Systems. Proc. of the 2nd DaWaK Conference (2000)
12. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. ADBIS'97 Symposium (1997)
13. Wojciechowski M., Zakrzewicz M.: Methods for Batch Processing of Data Mining Queries, Proc. of the 5th International Baltic Conference on Databases and Information Systems (2002)
14. Zaki M.J.: Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3 (2000)

taDOM: A Tailored Synchronization Concept with Tunable Lock Granularity for the DOM API

Michael P. Haustein and Theo Härder

University of Kaiserslautern
P.O. Box 3049, Kaiserslautern, Germany
{haustein,haerder}@informatik.uni-kl.de

Abstract. Storing, querying, and updating XML documents in multi-user environments requires data processing guarded by a transactional context to assure the well-known ACID properties, particularly with regard to isolate concurrent transactions.
In this paper, we introduce the taDOM tree, an extended data model which considers organization of both attribute nodes and node values in a new way and allows fine-grained lock acquisition for XML documents. For this reason, we design a tailored lock concept using a combination of node locks, navigation locks, and logical locks in order to synchronize concurrent accesses to XML documents via the DOM API. Our synchronization concept supports user-driven tunable lock granularity and lock escalation to reduce the frequency of lock requests both aiming at enhanced transaction throughput. Therefore, the taDOM tree and the related lock modes are adjusted to the specific properties of the DOM API.

1 Introduction

The use of the extensible markup language XML [1] for electronic data interchange leads to an enormous growth of the number and size of files keeping semi-structured XML data. In contrast, only structured data is managed by relational or object-relational database systems ((O)RDBMSs) so far. In order to allow combined processing of XML and relational data in an efficient way, it is indispensable to maintain XML documents in these database systems, too. On the other hand, (O)RDBMSs run transactions as their unit of control thereby guaranteeing the well-known ACID properties [2] which greatly improve the overall consistency, reliability and robustness of data management. As a consequence, managing XML documents by an (O)RDBMS also provides transactional properties for semi-structured data. Nevertheless, to accomplish effective and efficient processing for such documents, the control of their traversal and manipulation operations has to be adjusted to the transaction demands.

Different approaches to store XML documents in a relational database system and their performance characteristics are discussed in detail in [10], [11], and [12]. As a major issue, the ways XML documents are stored in a relational database lead to different synchronization problems. If an XML document is contained in a single CLOB attribute, locking may only take place at the document level. In contrast, if an XML document is shredded and stored across several tables, insertion of a new XML element may also affect a large part of the document (and even other documents). Insertion of a new XML element often results in several insert operations to relational database tables (depending on the shredding algorithm). Due to inadequate synchronization mechanisms, many database systems lock each of the affected tables entirely to prevent phantoms. Hence, such a crude method causes locking of document fragments by accident, even of unrelated documents which are shredded to those tables.

While efficient mechanisms are only available for the data side, combined processing of XML and relational data inside a relational database system requires such mechanisms also for the document side, that is, a native storage format with tailored lock mechanisms for XML documents in the first place.

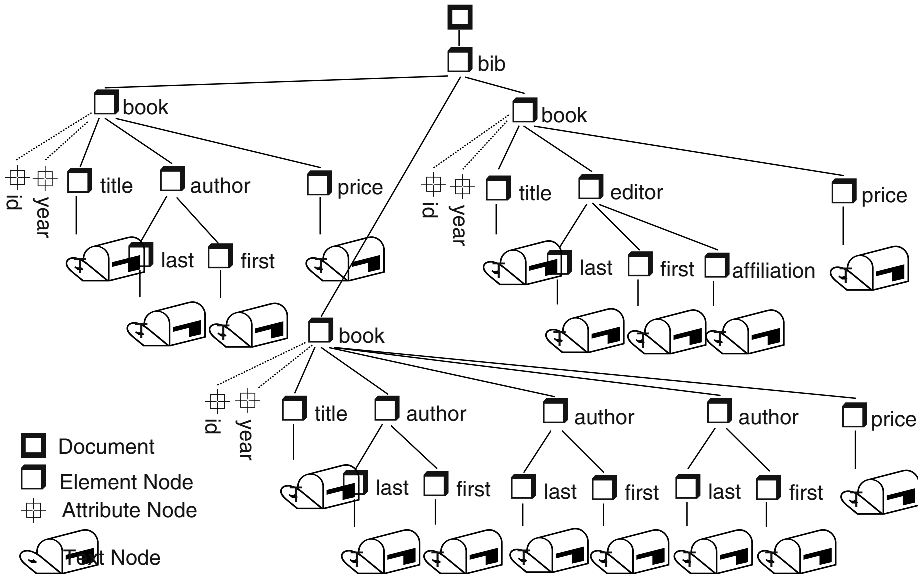


Fig. 1. A sample DOM tree

After having succeeded to store XML data within an (O)RDBMS in a native way equitable to relational data, we can immediately exploit the mature transactional concepts for atomicity, consistency, and durability. However, this is not true for transaction isolation. Since the structure of XML documents widely differs from the one of records and tables, we need a new concept to synchronize concurrent accesses to XML documents to provide for acceptable transaction performance.

Our primary objective is to develop a mechanism for concurrency control of semi-structured data where the properties of the document access interface are explicitly taken into account. For this reason, we refer to the DOM API [3] in order to query, traverse, and update XML documents stored in a database (see Section 2). In Section 3, we introduce the taDOM tree, a data model to represent XML documents. The structure and node types of the taDOM tree are specifically tailored to the properties of the DOM API. Based on the taDOM tree, we present in Section 4 an adjusted lock concept to synchronize concurrent accesses to XML data. Since the DOM API provides both, methods to traverse XML documents node-by-node and methods to apply simple queries, we use, in addition to node locks, a combination of navigation locks for the edges connecting the nodes, and logical locks in order to prevent phantoms.

Our concept supports tunable lock granularity and tunable lock escalation which can be independently specified for each document. Finally, in sections 5 and 6, we give a brief overview of related work about XML synchronization and wrap up with conclusions and some aspects of future work.

2 DOM API

The DOM API [3] provides a tree-based view to traverse and update XML documents. Elements and attributes of the document are represented as nodes of a directed acyclic graph (DAG). To illustrate the DOM tree, we consider a small XML fragment depicted in Figure 2, which is a modified version of the *bib.xml* document in [4]. The document contains a list of three books where each book contains the two attributes *year* and *id* and is described by *title*, *price*, *author*, and *editor*, respectively.

Figure 1 shows the corresponding DOM tree which represents the structure of the XML fragment defined in Figure 2. The outer element (in this case the <*bib*> element) is always assigned to a *document* node. Nested elements in the XML document are connected with edges in the DOM tree. Attributes are assigned to elements; they can be queried by their names. Attribute values and text values between opening and closing element tags in the document are stored in attribute nodes or text nodes (connected to the element nodes), respectively.

```
<bib>
   <book year="1994" id="1">
      <title>TCP/IP Illustrated</title>
      <author>
         <last>Stevens</last>
         <first>W.</first>
      </author>
      <price> 65.95</price>
   </book>
   <book year="2000" id="2">
      <title>Data on the Web</title>
      <author>
         <last>Abiteboul</last>
         <first>Serge</first>
      </author>
      <author>
         <last>Buneman</last>
         <first>Peter</first>
      </author>
      <author>
         <last>Suciu</last>
         <first>Dan</first>
      </author>
      <price>39.95</price>
   </book>
   <book year="1999" bid="3">
      <title>The Economics of...</title>
      <editor>
         <last>Gerbarg</last>
         <first>Darcy</first>
         <affiliation>CITI</affiliation>
      </editor>
      <price>129.95</price>
   </book>
</bib>
```

Fig. 2. Example of an XML fragment

A set of standardized methods allows the traversal of the graph node-by-node along edges, inserting or deleting nodes, as well as updating node values. Additionally, there exist some methods to apply simple queries to the XML document. For these reasons, we distinguish *navigational access*, *update access*, and *query access* when characterizing the methods of the DOM API.

Navigational access is provided by methods such as *getAttributes()*, *getFirstChild()*, *getNextSibling()*, or *getParentNode()*. The DOM API allows by invoking these and other methods to build a list of all attributes or all child nodes of a node, to navigate to the parent node, to the previous or next sibling node, and to the first or last child node. Methods such as *insertBefore(...)*, *removeChild(...)*, *setNodeValue(...)*, or *setAttribute (...)* support updating XML documents. These methods allow to insert new nodes, remove or replace child nodes, set new node values, and set or remove attributes via the DOM API.

Furthermore, simple queries for XML attributes or XML elements are provided by methods such as *getElementById(...)*, *getElementsByTagName(...)*, or *hasAttribute(...)*. Hence, an element referenced by an ID attribute or a set of elements qualified by their name can be queried.

When the DOM API is initialized for an XML document, the document is parsed and the DOM tree for the entire document is composed in main memory. While this approach provides for fast data access (after initialization), it is a very memory-consuming solution. To give a hint for the required order of magnitude, an XML document of 20 MB may consume up to 400 MB main memory [14]. In order to reduce this enormous memory consumption, some parsers ([15]) offer the option *deferred DOM*, a processing mode which loads the required nodes into memory on demand.

The current version of the DOM API is designed for single user environments. If multiple users open the same XML file, a local DOM tree is constructed for each user, leading to a potentially high degree of replication. On the other hand, a local DOM tree is not affected by modifications of its replicas, even if their update operations are in the committed state. Modifications can only be propagated to the stored XML document by simply overwriting the entire file. Of course, this proceeding leads to the loss of all updates which have been performed by concurrent users, even if their updates only affect non-overlapping sub-trees.

In contrast, *Persistent DOM* from Infonyte-DB ([16]) maps the DOM tree to a compressed binary file representation and provides concurrent access to the file with guaranteed atomicity and durability properties, but no support for the isolation and, in turn, for the consistency aspects. Any updates (committed or not) are immediately visible to all concurrent users which makes the individual user responsible for his view to the document and for the overall consistency of the XML file.

To improve this highly undesirable situation, a centralized and application-independent isolation mechanism is mandatory for the DOM API. First of all, we need a data model preserving the DOM interface properties, but allows for efficient lock acquisition and concurrent operations in addition. For this purpose, we introduce a suitable data model in the form of the taDOM tree in the next section.

3 taDOM Tree

Synchronizing concurrent accesses to XML documents calls for an appropriate logical representation of XML documents which supports fine-grained lock acquisition. To this end, we have developed the taDOM tree, an extension of the XML tree representation of the DOM (see Section 2). To illustrate its construction principles, we again consider the small XML fragment depicted in Figure 2.

Figure 3 shows the corresponding taDOM tree which represents the structure of the XML fragment. In order to construct such a taDOM tree, we introduce two new node types: *attribute root* and *string*. These new node types only appear within the taDOM tree—the logical representation of the XML document.
As far as synchronization is concerned, locks on the taDOM tree are always acquired transparently by a lock manager component introduced in Section 4. This proceeding guarantees that the behavior of the DOM API will not change from the users point of view.

The top of the taDOM tree consists of the document node, followed by the <*bib*> element—the outer element of the document. For such a document node, a root lock can be acquired which controls access to the entire document or arbitrary parts.

In contrast to a DOM tree in Figure 1, attribute nodes are not directly connected to the element nodes. Instead an attribute root is connected to each element and organizes the attributes of the corresponding element as descendant nodes. Such attribute roots are introduced to support synchronization of the *getAttributes()* method in an efficient way. In order to build a (frequently requested) *NamedNodeMap* containing all attributes of an element, only a single lock has to be acquired for the attribute root thereby enabling shared access to all attributes. Section 4.1 gives a detailed example for that procedure.

In the taDOM tree, XML elements are represented by element nodes and XML attributes by attribute nodes. The text within elements is handled by text nodes whereas the actual values of text nodes or attribute nodes are stored in string nodes. The existence of a string node is hidden to the taDOM API users. The users read or update values as usual by method invocations on the attribute or text nodes, but reading or updating a node value causes an implicit lock acquisition on the corresponding string node. This proceeding makes sense, because the DOM API request for an element or attribute does only return an element node or an attribute node, but not the value of such an node. To get the actual value of a node, the method *getValue()* or *getData()* has to be invoked. Only when such a method is executed, the actual node values (string nodes) have to be locked.

When a taDOM tree is constructed in the way described, lock granularity can be provided even for string nodes and, therefore, our concept enables *maximal concurrency* for DOM API accesses. An example for maximal concurrency is the following. Assume transaction T_1 builds an attribute list and reads all attributes of the first <*book*> element in Figure 3. Then T_1 requests the value of the attribute *id*. Note, although transaction T_1 already operates at the attribute value level, another transaction T_2 should be able to update the value of the attribute *year* in the same element. This access should be possible, because, so far, T_1 only has information about the existence of all attributes and not about all their values.

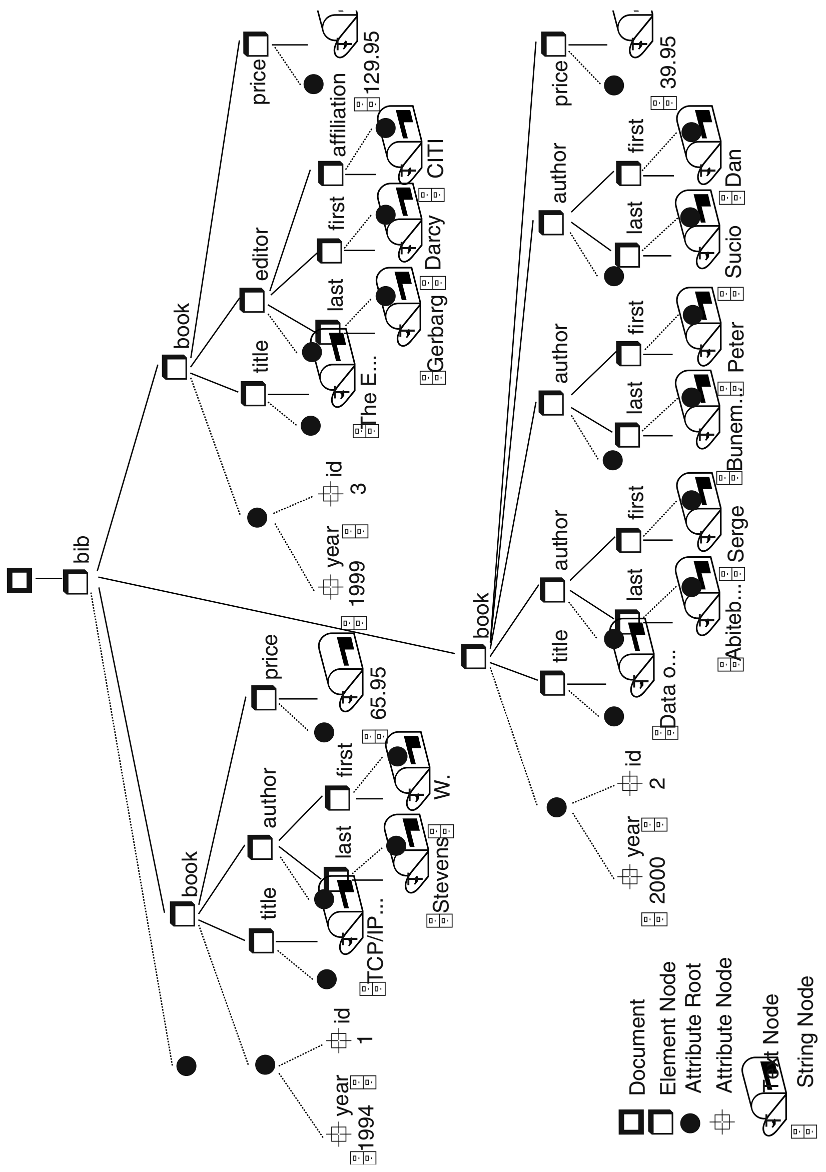


Fig. 3. A sample taDOM tree

4 Synchronization of Document Access

So far, we have only explained the newly introduced node types and how individual access to them can be improved. In a concurrent environment, all accesses to XML documents via the DOM API have to be synchronized using appropriate protocols. To facilitate our approach, we classify the related access methods in *navigational access*, *update access*, and *query access* (see Section 2).

In all cases, accessing a node within the taDOM tree requires the acquisition of a lock for the corresponding node. This procedure is described in Section 4.1. Maintaining locks, granting and declining lock requests have to be managed by a lock manager component. This component is also responsible for lock granularity and lock escalation management considered in Section 4.2.

When an XML document has to be traversed by means of navigational methods, the use of the navigation paths needs strict synchronization. This means, a sequence of navigational method calls must always obtain the same sequence of result nodes. To support this demand, we present so-called *navigation locks* in Section 4.3. Furthermore, query access methods also need strict synchronization to accomplish the well-known repeatable read property. For example, the method *getElementsByTagName()* should always yield the same node list within a transactional context. This means the insertion of a new element node with a specified tagname must be blocked if a concurrent transaction has already built a node list containing all existing nodes with this tagname. This demand and the existence of update methods can be facilitated by logical locks which are explained in Section 4.4.

4.1 Node Locks

	-	IX	NR	CX	LR	SR	U	X
IX	+	+	+	+	+	-	-	-
NR	+	+	+	+	+	+	-	-
CX	+	+	+	+	-	-	-	-
LR	+	+	+	-	+	+	-	-
SR	+	-	+	-	+	+	-	-
U	+	+	+	+	+	+	-	-
X	+	-	-	-	-	-	-	-

Fig. 4. Compatibility matrix for lock modes

While traversing or modifying an XML document, a transaction has to acquire a lock for each node before accessing it. The currently accessed node will be called *working node* in the following. Its lock mode depends on the type of access to be performed. Since the DOM API not only supports navigation starting from the document root, but also allows jumps „out of the blue“ to an arbitrary node within the document, locks must be acquired in either case for the sequence of nodes from the document root downwards to the working node. According to the principles of multi-granularity (or hierarchical) locking schemes, such intention locks communicate a transaction‘s processing needs to concurrent transactions. In particular, they prevent that a sub-tree S can be locked in a mode incompatible to locks already granted to S or sub-trees of S. Figure 4 gives an overview of the compatibility matrix for the lock modes defined as an extension of the well-known DAG locking ([5]). The effects of the different lock modes can be sketched as follows:

- The NR mode (node read) is requested for read access to the working node. Therefore, for each node from the document root downwards to the working node, a NR lock has to be acquired. Note, the NR mode only locks the specified node, but not any descendant nodes.
- The LR mode (level read) locks an entire level of nodes in the taDOM tree for shared access. For example, the method *getChildNodes()*, called for the working node, only requires an LR lock on the working node and not a number of single NR locks for all child nodes. In the same way, an LR lock, requested for an attribute root, locks all attributes for the *getAttributes()* method.
- To access an entire sub-tree of nodes (specified by the working node as the sub-tree root) in read mode, the SR mode (sub-tree read) is requested for the working node. In that case, the entire sub-tree is granted for shared access. Since the sub-tree can be determined by the user, this lock mode enables flexible concurrency control with tunable granularity.
- To modify the working node (updating contents or deleting the entire node with its sub-tree), an X lock (exclusive) is acquired for the working node. It implies the request of a CX lock (child exclusive) for its parent node and an IX lock (intent exclusive) for its ancestor nodes up to the document node. The CX lock for a node indicates the existence of an X lock for an arbitrary direct-child node whereas the IX lock indicates an X lock for a node located somewhere in the sub-tree. This varying behavior of the CX and IX locks is achieved by the compatibility of the IX and LR locks, and the incompatibility of the CX and LR locks.
- A U lock supports read with (potential) subsequent write access and prevents granting further R locks for the same object. A U lock can be converted to an X lock after the release of the existing read locks or back to an R lock if no update action is needed. It solves the problem of using a too restrictive mode when the necessity of write access is value-dependent.

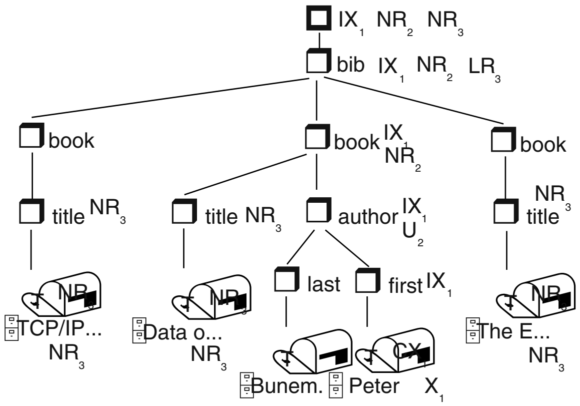


Fig. 5. Node locks on the taDOM tree

Figure 5 shows a cutout of the taDOM tree depicted in Figure 3 and illustrates the result of lock request sequences for the following example: Transaction T_1 starts modifying the value *Peter* and, therefore, acquires an X lock for the corresponding string node. This action implies the acquisition of the CX and IX locks for all preceding nodes up to the root by the lock manager component. Simultaneously, transaction T_2 intends to delete the entire <*author*> node which includes the string *Peter*. Therefore, T_2 has to acquire an X lock for the corresponding <*author*> node. This request, however, is declined because of the existing IX lock of T_1 and results in the request for a U lock instead in order to prevent further locks of read transactions. Meanwhile, transaction T_3 generates a list of all book titles and requests an LR lock for the <*bib*> node to obtain read access to all child nodes. Note, LR enables access to all direct child nodes without the need to perform a lock acquisition operation for each child. Then for each <*book*> node, the paths downwards to the title strings are locked by means of NR locks.

4.2 Tunable Node Lock Granularity and Lock Escalation

Both the SR lock, which locks an entire sub-tree in the taDOM tree in shared mode, and the X lock, which locks an entire sub-tree exclusively, enable tunable lock granularity and lock escalation for shared and exclusive locks, respectively. The combined use of them improves operational throughput because, due to lock escalation, the number of lock requests can be reduced enormously and, due to fine-tuned lock granularity, transaction concurrency may be increased.

To tune the lock granularity of nodes for each document, the parameter *lock depth* is introduced. Lock depth describes the lock granularity by means of the number of node levels (starting from the document node) on which locks are to be held. If a lock is requested for a node whose path length to the document root is greater than the lock depth, only a SR lock for the ancestor node belonging to the lock-depth level is requested. In this way, nodes at deeper levels than indicated by lock depth are locked in shared mode using SR locks on nodes at the lock-depth level. This allows a transaction to traverse a large document fragment in read mode without acquiring any additional node locks. In the same way, several X locks can be merged to a single X lock at a higher level.

I Figure 6 shows the locked taDOM tree cutout of Figure 5 where the effect of the lock-depth parameter is illustrated. With lock depth equal to 3, the NR locks of transaction T_3 beneath the <*title*> nodes are replaced by SR locks for the <*title*> nodes because the <*title*> nodes reside in node level 3 in this example. The IX, CX, and X locks of transaction T_1 beneath the <*author*> node are replaced by an X lock for the <*author*> node. As a prerequisite, this X lock requires a CX lock for the ancestor <*book*> node and IX locks for the <*bib*> node and the document root as described by the acquisition protocol of X locks in Section 4.1.

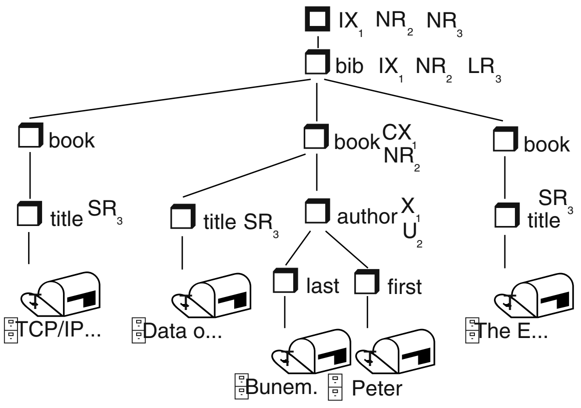
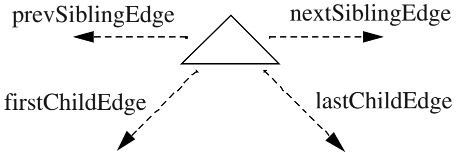


Fig. 6. Coarse-grained node locks with lock depth 3

In a similar way, lock escalation can be realized. To tune the lock escalation, we introduce two parameters, the *escalation threshold* and the *escalation depth*. The lock manager component scans the taDOM tree at prespecified intervals. If the manager detects a sub-tree in which the number of locked nodes of a transaction exceeds the percentage threshold value defined by the escalation threshold, the locks held are replaced with a suitable lock at the sub-tree root, if possible. Read and write locks are replaced by SR and X locks, respectively. This replacement procedure is the same as described above for the tunable lock granularity. The escalation depth defines the maximal sub-tree depth from the leaves of a taDOM tree to the scanned sub-tree root.

Obviously, there is certainly a trade-off to be observed for lock escalation. On the one hand, lock escalation decreases concurrency of read and write transactions, but, on the other hand, a reduction of the number of held locks and of lock acquisition operations is achieved saving lock management overhead.

4.3 Navigation Locks



	-	ER	EU	EX
ER	+	+	-	-
EU	+	+	-	-
EX	+	-	-	-

Fig. 7. Virtual navigation edges and compatibility matrix for navigation locks

So far, we have discussed optimization issues for locks where the object to be accessed was specified in some declarative way (for example, with a key value or a predicate). In addition, the DOM API also provides for methods which enable the traversal of XML documents where the access is specified relative to the working node (see Section 2). In such cases, synchronizing a navigation path means that a sequence of navigational method calls or modification (IUD) operations—starting at a known node within the taDOM tree—must always yield the same sequence of result nodes within a transaction. Hence, a path of nodes within the document evaluated by a transaction must be protected against modifications of concurrent transactions. For this reason, we introduce *virtual navigation edges* and corresponding *navigation locks* for element and text nodes within the taDOM tree.

While navigating through an XML document and traversing the navigation edges, a transaction has to request a lock for each edge. To support such traversals efficiently, we offer the ER, EU, and EX lock modes which correspond to the well-known R/U/X locks for relational records or tables ([5], [6]). Their use observing the compatibilities shown in Figure 7 can be summarized as follows:

- An ER lock (edge read) is acquired, before an edge is traversed in read mode. For example, such an acquisition may happen by calling the *getNextSibling()* or *getFirstChildNode()* method for the nextSiblingEdge or the firstChildEdge, respectively.

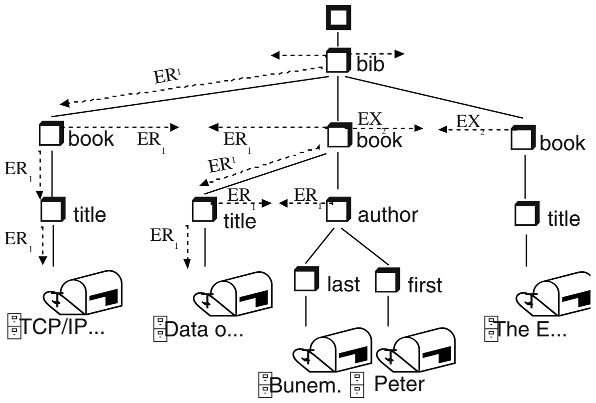


Fig. 8. Navigation locks on the taDOM tree

- An EX lock (edge exclusive) is requested, before an edge is modified. It may occur when nodes are deleted or inserted. For all edges, affected by the modification operation, EX locks are acquired, before the navigation edges are redirected to their new target nodes.
- The EU lock for edge updates eases the starvation problem of write transactions as described in Section 4.1.

Figure 8 illustrates an example where navigation locks on virtual navigation edges are acquired. Transaction T_1 starts at the <*bib*> node and reads three times the first child node (this means the node sequence <*bib*>, <*book*>, <*title*>, <*text*>) in order to get the string value of the first book title. Then T_1 determines the next sibling node of the current <*book*> node and repeats twice the first-child method to get the title of the second book. Now for this example the requested book is located, and T_1 finally gets the next sibling of the current <*title*> node which is an <*author*> node. As you can see, the synchronization concept allows another transaction T_2 to concurrently insert a new book by acquiring two EX locks for the sibling edges of the last two <*book*> nodes.

4.4 Logical Locks

The DOM API also provides the methods *getElementsByTagName(...)* and *getElementById(...)* to get elements by their tag names or by an Id attribute. The method *hasAttribute(...)* checks the existence of an attribute specified by its name. Read access to XML documents using these methods requires prevention of the phantom anomaly.

LocksTagnameQuery			
TAID	lock	scopeNID	tagname
...	R/U/X	...	...

LocksAttributeQuery			
TAID	lock	NID	attribute
...	R/U/X	...	...

LocksIDQuery		
TAID	lock	ID
...	R/U/X	...

Fig. 9. Lock tables for logical locks

Phantoms confusing read transactions may occur by insertions of concurrent transactions. As feasible in hierarchical (multi-granularity) locking schemes, we can prevent phantoms in our hierarchical data model for XML documents by using coarser lock granules. A definite disadvantage, coarse lock granules can enhance lock conflicts or deadlock situations. Especially, the *getElementById(...)* method, which can be invoked only for the document node, would always cause a read lock for the entire document. Since this is highly undesirable, we introduce a refined concept – so-called logical locks – to prevent phantoms.

To extend our synchronization concept by logical locks we introduce three lock tables which maintain the requested locks as illustrated in Figure 9. While the effects of node locks and navigation locks could be conveniently demonstrated using taDOM tree representations, the situation here is more complex and requires three separate tables for its description. Lock compatibility inside each table is handled by the conventional R/U/X compatibility matrix.

Table *LocksTagnameQuery* maintains the logical locks for element name requests. Execution of the method *getElementsByTagName(...)* requires an R lock for the corresponding transaction, tag name, and node ID of the node the method was invoked for. This ensures that the result node set of that method call does not change, because insertion of new nodes requires an X locks for the corresponding tag names in table *LocksTagnameQuery*. Deletion of nodes is still protected by node locks introduced in Section 4.1.

Table *LocksAttributeQuery* synchronizes the execution of *hasAttribute(...)* queries. The result (true or false) must not change within a transaction. This requires the acquisition of an R lock for the corresponding queried attribute name before getting the result. The other way around, inserting new attributes requires an X lock for the new attribute name in table *LocksAttributeQuery*. Also deletion of an attribute requires an X lock for the attribute name, because the *hasAttribute(...)* method just returns the result true or false and does not call for an R lock on the corresponding attribute node.

Table *LocksIDQuery* maintains logical locks for ID attributes. Since search for ID attributes is only supported at the document level, we do not consider a scope node ID in that lock table. In the same manner as in table *LocksAttributeQuery*, searching, inserting, and deleting of attributes requires R or X locks, respectively.

LocksTagnameQuery

TAID	lock	scopeNID	tagname
2	R	4711	last

LocksIDQuery

TAID	lock	ID
1	R	2
1	R	4

Fig. 10. Sample content of logical lock tables

Insertion of sub-trees (specified by the root node) in the taDOM tree requires the traversal of the entire sub-tree, before the actual insert operation can be performed. For each element found in the sub-tree, an X lock has to be acquired in table *LocksTagnameQuery*, and, for each ID attribute, an X lock in table *LocksIDQuery*. This is necessary, because the insertion of new elements and attributes can violate the repeatable read property for result node lists which have been filled with the *getElementById(...)* or *getElementByTagName(...)* methods.

Figure 10 illustrates some example contents of the logical lock tables. We consider again the XML fragment of Figure 2. At first, transaction T_1 is searching for an element with ID=2, then searching for an element with ID=4. Transaction T_2 is searching for all <*last*> nodes downwards from a <*book*> node with an assumed node ID 4711.

After having acquired the locks, another transaction T_3 is blocked when it attempts to insert a new node with ID=4 or to add a new <*author*> node with included <*last*> node to the <*book*> node with node ID 4711, because such an operation requires requesting X locks which cannot be granted. The insertion of a new <*last*> node is also forbidden in the entire sub-tree with root node ID 4711.

This example points up that lock compatibility also has to regard overlapping scopes. This means that inserting a new node requires acquisition of logical X locks for the tag name of the new node in table *LocksTagnameQuery* for each node from the document node downwards to the parent node of the new node. The deletion of existing nodes is already synchronized with read locks held by transactions accessing the nodes.

5 Related Work

So far, there are relatively few publications on synchronizing query and update access to XML documents. In [9], transactional synchronization for XML data in client/server web applications is realized by means of a check-in/check-out concept. User-defined XML fragments of stored XML documents are checked out and processed at the client side where the read and write sets of the operations are logged. Before invoking the check-in procedure, read and write sets are validated against the original document stored in the server. Upon success, the changes are propagated, whereas upon failure, the client has to decide whether to discard its own changes or to overwrite the changes of other clients. Lam et al. [13] discuss synchronization for mobile XML data with respect to handheld clients which query XML fragments by XQL. The authors present an efficient mechanism to determine whether two XQL expressions overlap. If a conflict is detected, a resolution algorithm is applied. Hehner et al. [8] discuss, also based on the DOM API, several isolation protocols (both pessimistic and optimistic) for XML databases. Node locks are not acquired in a hierarchical context, and lock granularity is fixed for each protocol. The ID lookup problem (*getElementById()* method) as well as jumps to arbitrary nodes within an XML document are not discussed in detail. XMLTM, an efficient transaction management for XML documents, is presented in [7] where XML documents are stored in relational databases having special XML extensions and an additional transaction manager. The authors also apply a combination of hierarchical lock acquisition and logical locks. Since XMLTM does not support a specific query interface, locks to synchronize navigational operations are not coped with. They do not consider varying lock granularities, but consider reducing the degree of isolation. Performance measurements with an implementation on IBM DB2 using the XML extender are presented. Both [7] and [8] do not treat attribute nodes in a special way or discuss lock escalation to increase data throughput.

6 Conclusion and Future Work

Synchronizing concurrent access to XML documents is an important practical problem. So far, only few extensions exist in commercial database systems which poorly support concurrent document processing primarily due to coarse-grained locking. On the other hand, concurrent XML processing has not received much attention in the scientific world yet.

In our paper, we have first presented the taDOM tree, an extended data model of the DOM representation of XML documents. The taDOM tree provides lock acquisition at very fine-grained levels. Based on the taDOM tree, we have introduced a concept of combined node locks, navigation locks, and logical locks (and their compatibilities) to synchronize concurrent access tailored to the DOM API. Node locks are acquired for the actual nodes of the taDOM tree, whereas navigation locks are acquired on virtual navigation edges to synchronize operations on the navigation paths. In addition, logical locks are introduced to prevent the phantom problem. The combination of the three lock types not only provides isolation for transaction processing with the DOM API, but also offers rich options to enhance transaction concurrency, for example, tunable lock granularity and lock escalation in order to increase throughput. Whether our concept is expressive enough to synchronize non-navigational APIs (like XQuery which works on node collections)—only by using node locks and logical locks—is part of future work.

Our next step is a system implementation in order to evaluate the synchronization concept and the locking behavior for different workloads. We plan to explore the effects on concurrency and performance for varying lock granularities and lock escalation thresholds. Future steps include the conjunction of XML synchronization with native XML storage methods and joined transactional processing of XML and relational data. Such a combined processing asks for the extension of the synchronization concept to provide the isolation property also for collection-based query languages like XQuery.

References

[1] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (2nd Ed.). W3C Recommendation (Oct. 2000)

[2] T. Härder and A. Reuter. Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys 15(4), (Dec. 1983) 287–317

[3] World Wide Web Consortium. Document Object Model (DOM) Level 2 Core Specification, Version 1. W3C Recommendation (Nov. 2000)

[4] World Wide Web Consortium. XML Query Use Cases. W3C Working Draft (Nov. 2002)

[5] J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc. (1993)

[6] T. Härder and E. Rahm. Database Systems – Concepts and Techniques of Implementation. (in German) Springer (2001)

[7] T. Grabs, K. Böhm and H.-J. Schek. XMLTM: Efficient Transaction Management for XML Documents. Proc. 11th Int. Conference on Information and Knowledge Management, McLean, Virginia, USA (Nov. 2002) 142–152

[8] S. Helmer, C.-C. Kanne, and G. Moerkotte. Isolation in XML Bases. Int. Report, Faculty of Mathematics and Comp. Science, Univ. of Mannheim, Germany, Volume 15 (2001)

[9] S. Böttcher and A. Türling. Transaction Synchronization for XML Data in Client-Server Web Applications. Proc. Workshop on Web Databases, Annual Conf. of the German and Austrian Computer Societies, Vienna (2001) 388–395

[10] D. Florescu, D. Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Rapport de Recherche, No. 3680, INRIA, Rocquencourt, France (1999)

[11] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. Proc. ACM SIGMOD, Madison, Wisconsin (June 2002) 204–215

[12] M. Yoshikawa and T. Amagasa. XRel—A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases. ACM Transactions on Internet Technology 1(1), (Aug. 2001) 110–141

[13] F. Lam, N. Lam, and R. Wong. Efficient Synchronization for Mobile XML Data. Proc. 11th Int. Conference on Information and Knowledge Management, McLean, Virginia, USA (Nov. 2002) 153–160

[14] T. Tesch, P. Fankhauser, and T. Weitzel. Scalable processing of XML with Infonyte-DB. Wirtschaftsinformatik-Zeitschrift 44(5) (in German), (2002) 469–475

[15] The Apache XML Project. Xerces2 Java Parser. http://xml.apache.org/xerces2-j/index.html.

[16] Infonyte-DB Version 3.0.0. User Manual and Programmers Guide. http://www.infonyte.com.

Client-Side Dynamic Preprocessing of Transactions[*]

Steffen Jurk[1] and Mattis Neiling[2]

[1] Brandenburg University of Technology
Cottbus, Germany
sj@informatik.tu-cottbus.de
[2] Free University
Berlin, Germany
mneiling@wiwiss.fu-berlin.de

Abstract. In a client-server relational database system the response time and server throughput can be improved by outsourcing workload to clients. As extention of client-side caching techniques, we propose to preprocess database transactions at the client-side. A client operates on secondary data and supports only a low degree of isolation. The main objective is to provide a framework where the amount of preprocessing at clients is variable and adapts dynamically at run-time. Thereby, the overall goal is to maximize the systems performance, e.g. response time and throughput. We make use of a two-phase transaction protocol that verifies and reprocesses client computations if necessary. By using execution statistics we show how the amount of preprocessing can be partially predicted for each client. Within an experiment we show the correspondence between amount of preprocessing, update frequency and response time.

1 Introduction

In client-server databases systems clients are more and more able to perform intensive computations on their own. On the one hand response time and server throughput can be improved by outsourcing workload to clients. On the other hand increasing demands on mobility require from clients to perform autonomous actions locally.

In the area of relational databases, several techniques provide client-side caching [15,14,2,8]. Most of these techniques address the situation where clients perform `SELECT-FROM-WHERE` SQL statements. Then the idea is to evaluate (preprocess) the query as much as possible on local cached data. A few of these techniques [2,8] examine the effect of transaction consistency of local execution of associated queries which requires extended isolation levels at the client side. However, transactions are still executed as a whole, whereby it might be more efficient and desirable to execute only a part of the transaction at the client.

[*] This research has been supported by the DFG, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316).

In the area of mobile data applications where clients might disconnect and move between different access points, the goal is to support a maximum of local autonomy. Autonomous mobile operations are proposed by [13] to enable updates on local data objects. However, it can suffer from heavy reprocessing [10] and can lead to unstable behavior as the workload scales up. In [6] a two-tier replication algorithm is proposed that allows mobile applications to perform tentative update transactions that are later on applied to a master copy. Many approaches have investigated the concept of local and global transactions, where local transactions are performed at the client side. An open-nested model [4] is used to represent transactions as a set of sub-transactions. In [12,11] weak (local) and strict (global) transactions are proposed to support dynamic object clustering. Semantic-based models [13] exploiting object semantics to facilitate autonomous and disconnected operations in mobile database applications.

In this work, we propose an alternative technique that aims at *client-side dynamic preprocessing of transactions*. The main objective is to provide a framework where the amount of preprocessing at clients is variable and adapts dynamically at run-time. Thereby, the overall goal is to maximize the systems performance, e.g. response time and throughput and to set up the amount of preprocessing for all clients accordingly. To the best of our knowledge this has not been addressed until now. The proposed technique extends client-side caching techniques by preprocessing transactions and provides a means for flexible preprocessing in mobile database applications.

We understand the preprocessing of transactions as sequencing them into two parts. Then the first part is executed at the client on secondary data and the second at the server on primary data. Secondary data is updated periodically by primary data. We motivate a preprocessing technique that is based on breakpoints that allows to handle imperative database update languages, such as PL-SQL, Transact-SQL (Section 3). To give maximum of autonomy to clients, they are not involved in concurrency control which requires to verify preprocessed transactions at the server side and possibly to reprocess them.

The amount of preprocessing is dynamic, adapts at run-time and depends on each individual client. It is useful due to the following reasons: (1) Clients access only of a subset of data and perform only some of the possible database requests. (2) Clients change their behavior. (3) New applications are used on top of the database. (4) Hard- and software improves over time which results into more computational power, more storage capacity, increasing network speed, etc. (5) Network topology changes.

The performance of a system depends on many factors (e.g. amount of data, user behavior, etc.). Our technique allows to use the amount of preprocessing and the mode of updating secondary data as run-time configurable parameters. Then the most challenging question is 'how much' preprocessing and 'how much' updating results into the best performance? We tackle the problem by using a real-valued performance measure P that is based on execution statistics (Section 4). The overall performance w.r.t. the configuration space S can then be characterized by a function $f : S \mapsto \mathbb{R}$ which maps each configuration to a performance value. In order to avoid unnecessary reconfigurations of the system, it is desired to identify the best configuration x_{opt} ($f(x_{opt})$ = minimal) by observing the systems behavior under the current configuration x. In this paper, we show (Section 5) that the performance behavior can be predicted for a subspace of S.

To show the correspondence between amount of preprocessing, update frequency and performance we present in Section 6 the results of our experiment.

2 Overview on the Architecture

In a client-server relation database system the server provides its functionality to clients via an interface. For simplicity we assume a finite number of database methods m_i that are written in a database update language, such as Transact SQL, PL-SQL, etc. Client are extended by a preprocessing unit as depict at Figure 1. In the following we refer to extended clients only.

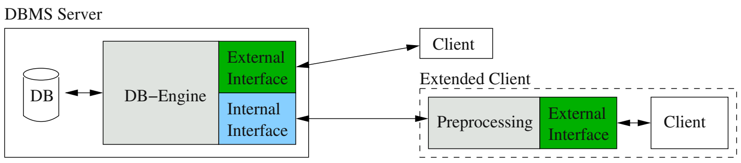


Fig. 1. Advanced client including a preprocessing unit.

A client k initiates m_i and preprocesses m_i. The preprocessing is defined by the notion of $cut_{k,i}$ which for each client k and method m_i combination defines the amount of preprocessing. The client connects to the server and exchanges information about preprocessing of m_i via some internal interface. The server verifies the preprocessing, initiates reprocessing if necessary and completes the execution of m_i (post-processing). Finally the result is passed back to the client. For more details can be found in Section 3.

Data at the client are secondary and updated periodically by the server. The amount of secondary data is determined by the amount of preprocessing. For a client k we denote its data set D_k which is a set of tables. The synchronization frequency for a client k and a table r is denoted as $f_{k,r}$ ($f_{k,r} = 0$ means no synchronization). Note that by increasing the update-frequencies $f_{k,i}$ the up-to-dateness of secondary data increases and the amount of reprocessing decreases.

Synchronization poses problems in guaranteeing consistency over all replicas. Existing approaches range from *pessimistic* (*eager*) to *optimistic* (*lazy*), see [6,16]. Pessimistic algorithms insist on a single-copy semantics to the user and update all replicas at once by using locking mechanisms to avoid conflicts. Optimistic algorithms manipulate replicated data with controlled inconsistencies. The algorithms are classified by three criteria: (1) single or multi-master systems, (2) log-transfer or content-transfer, and (3) push or pull based propagation. We follow a single-master, content-transfer strategy where data is pushed from the server to clients periodically.

The configuration of clients is handled by the server. For k clients, i methods and r tables, there are $k \cdot i + k \cdot r$ parameters corresponding to cuts $cut_{k,i}$ and update-frequencies in $f_{k,r}$. Each parameter is bounded by a finite set of possible values. A configuration x is a vector of all those parameters, each having a concrete value. A single parameter in x is denoted by $x[cut_{k,i}]$ or $x[f_{k,r}]$. The system runs a configuration x and constantly observes the performance and tries to figure out more efficient configurations y. Once some y has been found the system is reconfigured (see Section 5).

3 Processing Two-Phase Transactions

This section describes how methods are prepared for preprocessing on clients and how the amount of preprocessing can be defined. Afterwards we introduce the verification technique and present the two-phase transaction protocol.

3.1 Preparing Database Methods for Preprocessing

The dynamic nature requires fast adaptation of clients. That is once the amount of preprocessing changes also the amount of data (number of tables) changes. To support efficient changes, the clients data schema is restricted to tables only. Additional schema constraints (e.g. primary and foreign keys, domain constrains, etc.) are not allowed and shifted to methods. As a consequence the coupling between tables is very low and they can be removed and added easily. Clearly, once the schema of clients is different from the server, methods m_i cannot simply be executed at clients. Therefore we derive at compile-time for every method m_i a method m_i' that captures the *full and flat* method (including all other actions performed by the DBMS, e.g. primitive constraints, rules, trigger, etc.). The derivation of methods m_i' is non-deterministic, since usually the database management system decides the order of which integrity constraints are processed. We only provide an example (1) and assume in the following that for each m_i some m_i' is available.

Example 1. Consider some m as an insert of a tuple (x, y) into a table $T = (X, Y)$ and a database schema with a primary key defined on the first column of T. Once m is executed the DBMS takes care on enforcing the constraint. For a schema without keys the resulting flat method m' of m for could be either:

1. `IF NOT EXISTS (SELECT * FROM` T `WHERE` $X = x$)
   `THEN INSERT INTO` T `VALUES` (x, y) or
2. `INSERT INTO` T `VALUES` (x, y);
   `IF (SELECT COUNT(*) FROM` T `WHERE` $X = x$) $\geq$ `2 THEN ROLLBACK`

Clearly, the resulting m' depends on the order of enforcing the primary key — before or after the insert on T. □

3.2 Specify Preprocessing and Datasets by Breakpoints

For a method m'_i the amount of preprocessing is controlled by breakpoints. The basic idea is: once the execution of a m'_i at a client reaches a certain breakpoint it is stopped and continued at the server. The intention of breakpoint we use is the same as used in debugging programs. By default, a breakpoint is added after each instruction, one at the beginning ($b_\top$) and one at the end ($b_\perp$) of m'_i. Breakpoints, except $b_\top$ and $b_\perp$ can be modified by the database administrator. A preprocessing until $b_\top$ means that there is no execution, while until $b_\perp$ denotes the full execution of m'_i.

Example 2. Consider the following method from an e-shop:

```
$b_\top$;
IF EXISTS (SELECT * FROM Customer WHERE Login=l AND Passwd=p) THEN
    $b_1$;SELECT * FROM ShoppingCart WHERE Login=l
ELSE
    $b_2$;UPDATE Failure SET Count=Count+1 WHERE Login=l;
$b_\perp$
```

with its 4 breakpoints $b_\top$,b_1,b_2,$b_\perp$. □

Breakpoints and executable code between them form a finite directed graph, called breakpoint graph B_i, with a $b_\top$ node (no incoming edges) and a $b_\perp$ node (no outgoing edges) representing the two artifically added breakpoints, see Figure 2. The preprocessing of m'_i at the client k is precisely defined by a *cut* of the breakpoint graph. A $\mathrm{cut}_{k,i}$ is a spanning $\top$-tree ($b_\top$ is root of the tree) over B_i, where each $(\top, b)$-path in $\mathrm{cut}_{k,i}$ (a path starting with the $\top$ and ending with b) corresponds to a minimal $(\top, b)$-path in B_i. The minimum requirement ensures the unnecessary repetitions of breakpoints in loops.

Since a branching in $\mathrm{cut}_{k,i}$ might reflect the same code (example 2, breakpoints $b_\top$,b_1,b_2), we additionally require that each node in $\mathrm{cut}_{k,i}$ has either zero or all of its direct successors (as defined in B_i).

Then m'_i is executed along the breakpoints of $\mathrm{cut}_{k,i}$. Once a leaf-node has been reached the execution stops. Based on *cuts* the dataset D_k of each client can be precisely determined. Each edge of the cut represents a piece of code that either reads, writes or does not access data. Then the dataset D_k is the set of tables that are read/written by that code.

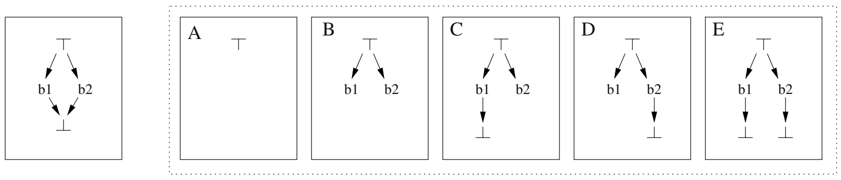


Fig. 2. Left: breakpoint graph; Right: all possible cuts

Example 3. The possible datasets w.r.t. cuts in Figure 2:
A — dataset: empty
B — dataset: `Customer`
C — dataset: `Customer, ShoppingCart`
D — dataset: `Customer, Failure`
E — dataset: `Customer, ShoppingCart, Failure` □

The execution of m'_i at k determines its own data space of interest and the correctness of the preprocessing depends on its up-to-dateness. Thereby each transaction requires its own specific up-to-dateness. Consider for example the following queries: all products of a certain category and the last 10 user logins. Clearly, from the behavior of such applications we know that secondary data for the first query need to be less up-to-date than those of the second.

Note that the breakpoint technique does not allow to split queries and to preprocess sub-queries. It can be improved, by adding a procedural representation of the query plan to m'_i that allows to add breakpoints into the query plan.

3.3 Verification of Preprocessed Transactions

The verification of a preprocessed method requires a fast and precise method. For this we use a combination of time stamps and techniques from *materialized view maintenance*, see [7,1,3]. As execution information, that is passed from clients to the server, we use the executed (preprocessed) part of the syntax tree of a method m'_i reflecting the executed code and the breakpoint path $b_\top, \ldots, b_j$. During the preprocessing, the syntax tree is attached by the result (value) of the evaluation of an expression of the update language (e.g. SQL, boolean, arithmetic, etc.) and the resulting updates on data. Further, the syntax tree contains timestamps of local tables that has been used for computation.

At the server, based on primary data, the verification procedure returns a path of breakpoints $b_\top, \ldots, b_{j'}$ with $j' \leq j$ that have been correctly preprocessed. In other words: whose execution would lead to the same result as executed directly on primary data. Ideally the preprocessing is correct and $j = j'$ holds, otherwise a reprocessing after $b_{j'}$ is necessary.

The basic idea is as follows: Follows up the syntax tree step-by-step and verify each step containing a SQL expression (the only read/write access on data). If client and server timestamps of used tables are equal the result of the expression must be correct. In case they are different we apply a finer test that is known as "query independent of update" or "irrelevant update". Then each SQL expression is considered as a materialized view over primary data, where the current content is based on secondary data. The problem left is to determine whether the content of the view is still valid. If so, the evaluation of the expression is consistent on primary data. We use an index structure that contains all updates applied on primary data during the last update period. Once the index contains a "relevant update" the content of the view is not up-to-date and therefore the verification of the SQL expression fails. Otherwise its must be correct. Note that only a subset of SQL expressions can be fully handled this way. Hence the verification might still reject steps although correctly preprocessed.

3.4 A Two-Phase Transaction Protocol

We shortly introduce the used two-phase transaction protocol that ensures the consistency of state while processing transactions. Note that step (2) to (4) are executed within a single transaction, since the result of the verification should not be spoiled by other actions executed in the meantime.

(1) Preprocessing. A client calls a method m_i and preprocesses the (extended) method m_i' according to $\text{cut}_{k,i}$. As soon as a leaf-node is reached the local execution stops. The execution does not modify secondary data. Any update is made temporally and not visible to other executions at the client. The client connects to the server and passes the execution information.

(2) Verification and Setup. The verification and the setup is done simultaneously. Based on the execution information, the idea is to set up the proper state of execution (e.g. assign local variables, apply preprocessed updates on primary data) for each verified breakpoint. Whenever an error has been detected the verification stops and returns the correctly preprocessed path $b_\top, \ldots, b_j$.

(3) Reprocessing and (4) Post-processing. After the verification the method is executed in the usual way. That is, each read and write access is done directly on primary data. Starting from the last successfully verified breakpoint b_j the method is reprocessed until a leaf node in $\text{cut}_{k,i}$ is reached which possibly follows another path in $\text{cut}_{k,i}$ than the preprocessing. Afterwards the method is post-processed until it commits.

(5) Exchange Result and Complete. The result of the method and a set of updates that has been applied on primary data during re- and post-processing of m_i' are sent back to the client and applied on secondary data. Further, temporally updates that occurred during the preprocessing are only applied on secondary data, if the corresponding edge of the preprocessed path has been successfully verified at the server.

4 Characterizing System Behavior and Performance

This sections introduces a log of executions and some useful measures on it. They are used afterwards to characterize and to predict the systems performance.

4.1 Execution Time and Distributions

The server maintains a log file *LOG* which contains an entry $\text{log}_{k,i}$ for each executed method m_i at client k. The following information is available for each single execution $\text{exec} \in \text{log}_{k,i}$:

1. $path(exec)$ - sequence of breakpoints marking the preprocessed path
2. $verified(exec)$ - sequence of breakpoints marking the correctly executed path
3. $re(exec)$ - sequence of breakpoints marking the path that is executed after $verified(exec)$, note that always $verified(exec) \subseteq path(exec)$ holds, whereas $path(exec) \cap re(exec)$ contains at least $b_\top$.
4. $t_{Pre}(exec)$ - the amount of time for preprocessing, including communication,
5. $t_V(exec)$ - the amount of time for verification,
6. $t_{Re}(exec)$ - the amount of time for reprocessing,
7. $t_{Post}(exec)$ - the amount of time for post-processing,
8. $t_{Pre}(exec, b_1, b_2)$ - the amount of time for preprocessing edge (b_1, b_2),
9. $t_V(exec, b_1, b_2)$ - the amount of time for verifying edge (b_1, b_2), and
10. $t_{Re}(exec, b_1, b_2)$ - the amount of time for reprocessing edge (b_1, b_2).

As a requirement of run-time adaptive systems, we observe the systems behavior only during the last time period T, where T is constant and has to be chosen adequately. The following notions refer only to entries in LOG within T.

The total number of executions is denoted as c, for a method m_i and a client k it is denoted as $c(k,i) =\mid log_{k,i} \mid$. For each edge (b_1, b_2) we derive from the logs the total number of preprocessings/verifications/reprocessings, denoted by $c_p(k, i, b_1, b_2)$ with $p \in \{Pre, V, Re\}$, and its corresponding average execution time

$$\bar{t}_p(k, i, b_1, b_2) = \frac{1}{c_p(k, i, b_1, b_2)} \sum_{exec \in log_{k,i}} t_p(exec, b_1, b_2)$$

The average execution time of a method m_i is defined as:

$$\bar{t}(k,i) = \frac{1}{c(k,i)} \cdot \sum_{exec \in log_{k,i}} \big(t_{Pre}(exec) + t_V(exec) + t_{Re}(exec) + t_{Post}(exec)\big)$$

Since it also contains extra time for communication, it reflects the average response time. In case there was no execution, $\bar{t}$ is defined to be 0. For a time analysis the standard deviation of the execution time, or the maximal execution time of a method might be of interest, and can be computed accordingly.

The frequency of an edge is characterized by the following distributions ($p \in \{\mathrm{Pre}, \mathrm{V}, \mathrm{Re}\}$).

$$D_p(k, i, b_1, b_2) = \frac{c_p(k, i, b_1, b_2)}{c(k,i)}$$

Note that D_{Pre} is a total distribution on the breakpoint branched tree of a method, since in any case one path is executed. The distributions of D_V and D_{Re} depend on the error rate. In case $c(k,i) = 0$ the distribution is 0.

4.2 Capturing Performance

For a method we consider a real-valued and time-sensitive performance measure $P(k, i) \in [0, \infty)$ where 0 denotes best and ∞ worst performance.

$$P(k,i) = \begin{cases} 0 & \bar{t}(k,i) \leq t_{bound} \\ (\bar{t}(k,i) - t_{bound})^2 & else \end{cases}$$

t_{bound} denotes a natural bound where performance does not matter. For example, a web application response time is excellent between 0 and 500 milliseconds. Above the the bound $t_{bound} = 500$ users might feel slower response time. Clearly, t_{bound} is application dependent. We use a quadratic function to put extra penalty on low performance. Based on the performance of method-client combinations we define the total performance which puts extra weights according to the total number of executions of k, i during T:

$$P = \sum_{k,i} \frac{c(k,i)}{c} \cdot P(k,i)$$

As stated earlier, the performance of the system depends on different factors, e.g. topology, network speed, client computation power, amount of conflicts between database actions, user and application behavior at access points, etc. The underlying system is responsible for guaranteeing optimal operating conditions by choosing an appropriate configuration x (amount of preprocessing and update frequency). Highlighting the dependence of some x and P, the above *LOG* is extended, such that each entry contains its corresponding configuration x. Further, we use the notion of P^x which refers only to such entries and represents the performance under the configuration x.

5 Identify the Right Amount of Preprocessing

The task of the server is to constantly monitor the overall performance and to identify *better* configurations y. Once such an y has been found the system is reconfigured. For a given x, the configuration space S can be split into two disjoint subsets $S_{est}(x)$ and $S_{try}(x)$. $S_{est}(x)$ denotes all those configurations y for which it is possible to estimate P^y by observing the systems behavior under x. $S_{try}(x)$ denotes those y for which there is no estimation possible.

5.1 The Estimated Neighborhood $S_{est}(x)$

Let $<$ denote a partial ordering over cuts of breakpoint graphs. For two cuts *cut*, *cut'* of a breakpoint graph B with $cut \neq cut'$ the property $cut' < cut$ holds, if *cut'* is a subtree of *cut*. Consider some method m_i', a client k and a configuration x. Its execution is specified by the parameter $\text{cut}_{k,i}$ and the resulting average response time is $\bar{t}^x(k,i)$. Let y be configurations that differ only in $\text{cut}_{k,i}$ to x, such that $y[\text{cut}_{k,i}] < x[\text{cut}_{k,i}]$. For all such y the costs $\bar{t}^y(k,i)$ can be estimated.

Let E be the set of edges where $x[\mathrm{cut}_{k,i}]$ and $y[\mathrm{cut}_{k,i}]$ differ. The resulting average execution time $\bar{t}^y(k,i)$ can be estimated as follows:

$$\begin{aligned}\bar{t}^y(k,i) \approx \bar{t}^x(k,i) &- \sum_{(a,b)\in E} D_{Pre}(k,i,a,b)\cdot \bar{t}_{Pre}(k,i,a,b)\\ &- \sum_{(a,b)\in E} D_V(k,i,a,b)\cdot \bar{t}_V(k,i,a,b)\\ &+ \sum_{(a,b)\in E} D_V(k,i,a,b)\cdot \bar{t}_{Re}(k,i,a,b)\end{aligned}$$

For each edge that is removed from $x[\mathrm{cut}_{k,i}]$ there will be no preprocessing at the client and no verification at the server. The current $\bar{t}^x(k,i)$ already contains an amount of reprocessing of $\sum D_{Re}(k,i,a,b)\cdot \bar{t}_{Re}(k,i,a,b)$ in case a preprocessed edge requires a reprocessing. By adding $\sum D_V(k,i,a,b)\cdot \bar{t}_{Re}(k,i,a,b)$ to $\bar{t}^x(k,i)$, $\bar{t}^y(k,i)$ considers all removed edges as 100% reprocessed which is equal to post-processing those edges at the server. In case $\bar{t}_{Re}(k,i,a,b)$ is unknown (e.g. the edge has not been reprocessed in T), its cost can be either taken from its last reprocessing (before T) or can be approximated by $\bar{t}_{Pre}(k,i,a,b)$, but the estimation in the latter two case is then accordingly imprecise.

5.2 Correcting Estimations

$S_{est}(x)$ contains all shortened configurations y (t.i. with earlier cuts). But changing from x to configurations y typically increases the workload of the server. The higher this workload, the more the estimation will be imprecise, since there is an upper bound of workload. To avoid wrong estimations we apply a continuous and smooth correction function $f_{cor} : \mathbb{R}^2 \to \mathbb{R}$ as follows. Let

$$w = \sum_{k,i}\ \sum_{exec\in log_{k,i}} t_V(exec) + t_{Re}(exec) + t_{Post}(exec)$$

be the current workload of the server, characterizing the total amount of computation time during T. Let further

$$w^+ = \sum_{k,i}\ \sum_{(a,b)\in E_{k,i}} c_V(k,i,a,b)\cdot \big(\bar{t}_{Re}(k,i,a,b) - \bar{t}_V(k,i,a,b)\big)$$

be the workload that would be added to the server by y ($E_{k,i}$ denotes the set of edges where $x[\mathrm{cut}_{k,i}]$ and $y[\mathrm{cut}_{k,i}]$ differ). Then the function f_{cor} computes for a given w and w^+ a correction factor $f_{cor}(w,w^+) \geq 0$ by which each of the estimated $\bar{t}_y(k,i)$ is corrected. The function f_{cor} is system specific and cannot be provided in advance. Hence, the system must learn the function. From load analysis we know about f_{cor}, that it is monotonous increasing. Hence the first and second partial derivatives has all to be positive, a simple example for such a f_{cor} is given by the quadratic function $1 + \alpha w^2 + \beta(w^+)^2$ with $0 < \alpha < \beta$.

5.3 A Search Strategy for Step-Wise Improvement

Since the performance P^x of some configuration x is not necessarily stable over time (due to changing user behavior, etc.), we apply a search algorithm that operates in parallel to the database computations and continuously looks for a better configuration. It operates as follows on x: (1) Extend the cuts in x to y and reconfigure, (2) Modify the frequencies in y to z and reconfigure (3) wait, (4) Search x_{min} in $S_{est}(z)$ with estimated P minimal, (5) Repeat step (2)-(4) or continue, (6) Pick best among x and all x_{min} and reconfigure, (7) wait, (8) Goto step (1).

Step (1): Extend cuts in x. Let E be the set of all tuples (k, i, a, b) of leaf-edges (edges that connect a leaf node) (a, b) for cuts $x[cut_{k,i}]$. Pick randomly some edges (k, i, a, b) and extend $x[cut_{k,i}]$ by all direct successors of b.

Step (2): Modify frequencies in y. Let E be a randomly chosen subset of edges (k, i, a, b) that are frequently preprocessed and frequently reprocessed. That is, $D_{Pre}(k, i, a, b) \cdot (D_{Pre}(k, i, a, b) - D_V(k, i, a, b))$ is high. Let further R_k be all tables that are required to preprocess $(k, i, a, b) \in E$ at the corresponding client k. Increase all update frequencies $y[F_{k,r}]$ with $r \in R_k$ by the next higher one.

Step (4): Search x_{min}. Based on the above estimated and corrected values $\bar{t}_x(k, i) \cdot f_{cor}(w, w^+)$ the performance of a method $P^x(k, i)$ and the overall performance P^x can be computed for each $x \in S_{est}$. Locating x_{min} in S_{est} is a typical optimization problem where a minimum has to be found for the function $f : S \to \mathbb{R}$ with $f(x) = P^x$. The problem can be solved by heuristic optimization techniques, such at *Tabu Search* [5] or *Simulated Annealing* [9].

The algorithm performs a random search on S_{try} and picks the best configuration out of S_{est} afterwards. The database administrator specifies the duration of (3), (7) and under which conditions (5) continues. Intuitively the duration of (7) should exceed those of (3), since (7) reflects the currently best known configuration. Note that our model does not consider reconfiguration costs yet.

6 Experimental Evaluation

In order to study the correspondence of the amount of preprocessing, update frequency and response time, a prototype of the proposed approach has been implemented in a DB2, MySQL, PHP environment.

The experimental setup is as follows: The DB2 database contains 20.000 objects and a single stored procedure that randomly reads and writes 20 of that objects. To achieve real load conditions, the server itself executes the stored procedure two times a second. The extended client keeps a copy of all objects in its MySQL database that is synchronized by the server in different time periods. Altogether we use 12 possible update frequencies f that delay the update of the client database by 0 up to 25 seconds. The meaning of 0 is that server updates are

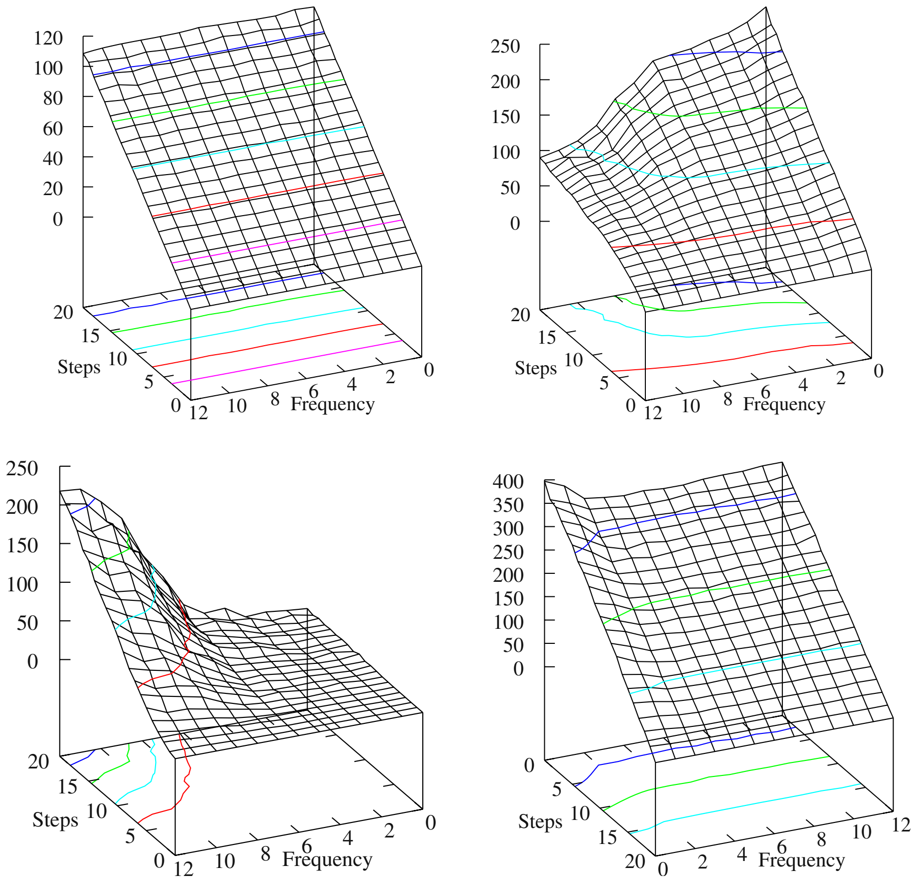


Fig. 3. From upper left to lower right: Average execution time of (1) preprocessing, (2) verification, (3) reprocessing and (4) post-processing in milliseconds.

immediately propagated to clients. Further, the client re-implements the stored procedure in PHP and executes i of the 20 steps locally while the remaining $20 - i$ steps are computed at the server.

For each combination of f and i we have measured the computation time of each phase of the transaction protocol and the total response time at the client. Thereby the client is performing one request (execution of the stored procedure) per second. Figure 3 depicts the computation time that is needed for the preprocessing, verification, reprocessing and post-processing phase.

The preprocessing increases linearly along the number of steps and update frequencies, since the load at the client is relatively low.

The verification increases along the number of steps, since more needs to be verified, and shows a different behavior for different update frequencies. It is maximal at $f = 0$ and $i = 20$, since the probability of an erroneous computation

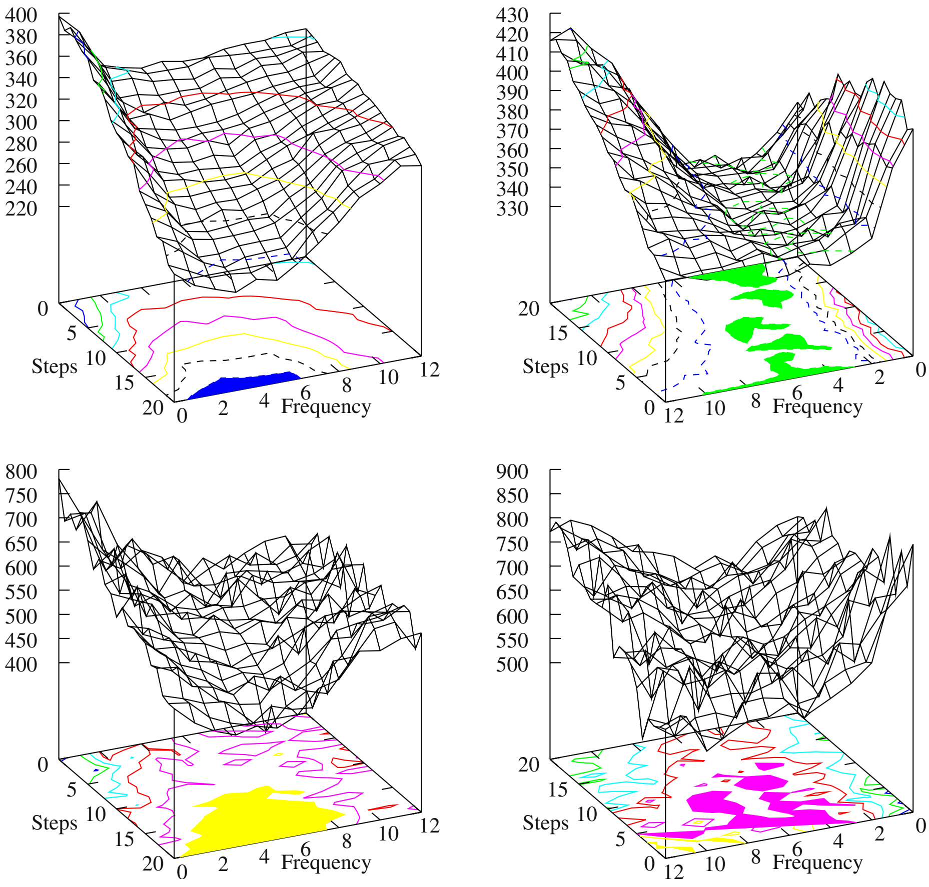


Fig. 4. Top left: Sum over verification, reprocessing and post-processing = server load. Top right: Total execution time for the above phases. Bottom: Total execution time for different setups, e.g. database size, load, etc. Minima are highlighted. Time in ms.

at the client is very low and a verification of all 20 steps is required. It is much lower at $f = 12$ and $i = 20$, since the probability of erroneous computations is higher and therefore less needs to be verified. Note that the verification technique stops as soon an errors has been detected.

According to the resulting error rates for low update frequencies, the reprocessing is maximal at $f = 12$ and $i = 20$. Note that the reprocessing time is very low up to an update frequency of $f = 6$.

The post-processing increases almost linearly along the number of steps $(20-i)$ that are computed at the server. Interesting to note is that there is a small peak at frequency 0 and 1. It results from the high synchronization overhead at the server, since updates are immediately (in case of $f = 0$) propagated to the client.

Figure 4 depicts the total computation time of the server (verification, re-processing and post-processing) and the total execution time (response time) for the above phases.

The optimal operating conditions (top left) can be achieved by using an update frequency between 5-7 (corresponding to an update delay of 2-4 seconds). By changing the setup, e.g. database size, client behavior, load, etc., the optimal operating conditions change as depicted by the lower diagrams of Figure 4. At the one hand they suggest to preprocess almost everything at the client, while on the other hand it is better to keep the computation at the server. Hence, the dynamics of such systems result into dynamic (non stable) optima.

In all the experiment has shown that the response time can be improved up to 35% compared to the situation where the client performs no preprocessing. However, the setup was rather artifical, since the behavior was modeled by random access. Currently, we simulate an e-shop Internet application where different users access product information, order products and maintain product lists and prices.

7 Conclusion and Future Work

In this paper we proposed *client-side dynamic preprocessing of transactions* as a performance-driven approach to distribute data and computation to clients. For a subspace $S_{est}(x) \subseteq S$ of the configuration space the systems performance can be predicted which allows us to automatically configure the preprocessing of clients.

The work on this subject has posed many new interesting problems that are currently under progress or part of our future work. These are: (1) Consideration of reconfiguration costs. (2) In this paper, we argued that there are full tables at client. Clearly, the use of table fragments is much more suitable. Further, by using event-driven updates instead of update frequencies the up-to-dateness of secondary data can be improved. (3) How to handle local rejects (rollback) on clients? (4) Currently, we randomly extend cuts and increase frequencies in order to find better configurations. We plan to provide heuristics by observing execution statistics at the whole breakpoint graph, not just on cuts. In detail we plan to apply *continuous dynamic optimization* techniques that are able to track moving optima. (5) A difficult problem is the configuration of the *wait* phases of the search algorithm and the definition of T. (6) The use of nested transactions allow client much more freedom in preprocessing, in case sub-transactions are independent. Are there design principles for preprocessing transactions, e.g. first checking of integrity, then updating? (7) The results of our e-shop simulation will be presented in a forthcoming paper.

References

1. Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. pages 254–263, 1998.
2. J. Basu and A. Keller. Degrees of transaction isolation in sql*cache: A predicate-based client-side caching system, 1996.
3. Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseak Shim. Optimizing queries with materialized views. In *11th Int. Conference on Data Engineering*, pages 190–200, Los Alamitos, CA, 1995. IEEE Computer Soc. Press.
4. Panos K. Chrysanthis. Transaction processing in mobile computing environment. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–83, Princeton, New Jersey, 1993.
5. Fred Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.
6. Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. pages 173–182, 1996.
7. Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques and applications. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, 1995.
8. Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal: Very Large Data Bases*, 5(1):35–47, 1996.
9. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
10. Sushil Jajodia Peng Liu, Paul Ammann. Incorporating transaction semantics to reduce reprocessing overhead in replicated mobile data applications, 1999.
11. E. Pitoura and G. Samaras. *Data Management for Mobile Computing*, volume 10. Kluwer Academic Publishers, 1998.
12. Evaggelia Pitoura and Bharat Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. In *15th International Conference on Distributed Computing Systems*, pages 404–413, 1995.
13. Gary D. Walborn and Panos K. Chrysanthis. Supporting Semantics-Based Transaction Processing in Mobile Database Applications. In *Symposium on Reliable Distributed Systems*, pages 31–40, 1995.
14. Y. Wang and L. A. Rowe. Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In *Proceedings of the 1991 ACM SIGMOD Int. Conf. on Management of Data*, pages 367–376. ACM Press, 1991.
15. W. K. Wilkinson and M. Neimat. Maintaining Consistency of Client-Cached Data. In *16th International Conf. on VLDB, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 122–133. Morgan Kaufmann, 1990.
16. Y. Saito. Optimistic Replication Algorithms, 2000.

Using Common Schemas for Information Extraction from Heterogeneous Web Catalogs

Richard Vlach[1] and Wassili Kazakos[2]

[1]Charles University, Malostranske nam 25, 118 00 Praha 1, Czech Republic
vlach@ksi.ms.mff.cuni.cz
[2] Forschungszentrum Informatik, Haid-und-Neu-Straße 10–14, 76131 Karlsruhe, Germany
kazakos@fzi.de

Abstract. The Web has become the world's largest information source. Unfortunately, the main success factor of the Web, the inherent principle of distribution and autonomy of the participants, is also its main problem. When trying to make this information machine processable, common structures and semantics have to be identified. The goal of information extraction (IE) is exactly this, to transform text into a structural format. In this paper, we present a novel approach for information extraction developed as part of the XI³ project. Central to our approach is the assumption that we can obtain a better understanding of a text fragment if we consider its integration into higher-level concepts by exploiting text fragments from different parts of a source. In addition to previous approaches, we offer higher expressiveness of the extraction schema and an advanced method to deal with ambiguous texts. Our approach provides a way to use one extraction schema for multiple sources.

1 Introduction

From the vast source of information provided by the World Wide Web, we are interested mainly in extracting information from Web catalogs. To build new applications integrating the content of several sources, we need to recognize the relevant information and restructure it according to our needs. Although the Web catalogs are nowadays largely in the form of semi-structured pages, mostly generated automatically from a database, there are no unified conventions of presenting the data. Typically, Web catalogues do not use grammatical structures, which prevent using natural language processing (NLP) techniques. Fortunately, many Web catalogs adopt a common, domain-specific basic terminology and reveal at least loose logical similarity in structuring presented information. This could be a key to coping with presentation heterogeneity within one domain.

Consider the example in Figure 1. Both (a) and (b) correspond to typical data accessible through the Web. Although the two representations differ significantly in the HTML code, they contain the same information. The goal is to transform the input text into a predefined fixed structure that is suitable for further processing. In the example, the interesting chunk of the target structure is shown in (c).

```
<li>ATX, AGP, 4x PCI</li>
```

(a)

```
<table>
 <tr><td>Expansion slot:</td>
     <td>1 x AGP<br>4 x PCI</td>
 </tr>
 <tr><td>Form factor:</td>
     <td>ATX</td>
 </tr>
</table>
```

(b)

```
<Motherboard>
   <AgpSlot/>
   <Slot>
      <SlotType>PCI</SlotType>
      <Amount>4</Amount>
   </Slot>
   <FormFactor>ATX</FormFactor>
</Motherboard>
```

(c)

Fig. 1. Example data (a), (b), and a target structure (c)

As a part of the XI³ project [7], we faced the problem of developing an extraction system that extracts information from heterogeneous Web sources within one domain, e.g. motherboard descriptions. As in the example above, the system must be able to process semi-structured as well as plain texts and to produce output according to a predefined structure. Important requirements were scalability in the number of distinct sources, possibility to obtain approximate results from new sources without a priori training, robustness in the case of meaningless input, and compliance with XML-based integration.

Obviously, the core part of the system is an extraction component. Many approaches to information extraction from the Web have been proposed. A survey of the area and introduction of key approaches and the best information extraction systems are given in [1] and [2]. An overview of finite-state approaches and relational learning techniques is introduced in [9]; other recent learning-based approaches to automate information extraction are presented in [8, 5]. In contrast to learned systems, knowledge-based systems are prepared by hand using knowledge of the application domain. These systems demonstrate better performance than the generated ones [1]. Additionally, the proposed learned wrappers are site specific, which means, an individual wrapper has to be built and trained for each site.

Two remarkable representatives of the knowledge-based approaches, which seem to fit our requirements, are the Jedi project [3] and an approach of Gao and Sterling [4]. We agree with Gao and Sterling [4] that despite diversity in data structures in a domain, regularities can be found in concept organization and concept descriptions, which are independent on the Web page layout. They can be observed also in the pure text representation of the Web pages, i.e., ignoring the HTML tags. The discovered regularities are consistent across Web sites as long as the sites are in the same domain. If we manage to represent the common regularities in a schema, it forms a means to guide information extraction over heterogeneous sources. Their approach uses a hybrid representation method for schemas of semi-structured data and provides a fast extraction algorithm that takes advantage of the semi-structured page layouts.

Jedi [3] provides a tool for the creation of wrappers to extract information from several independent information sources. It uses attributed context-free grammars, which are evaluated with a fault-tolerant parsing strategy to cope with ambiguous grammars and irregular sources.

Evaluating the above approaches, we found two areas in which they are insufficient with respect to our needs. First, the extraction schemas are not able to represent some important common regularities observed in sources; e.g. limited distance of logically related strings ([4]) or permutations on lower-level concepts ([3]). Second, both approaches lack a more sophisticated mechanism to solve any ambiguity of a given source.

Inspired by existing ideas, we have developed a new approach with one hand-crafted extraction schema to cover multiple sources, and an extraction algorithm that uses an advanced method to deal with ambiguous texts. The schema accommodates the source-independent structural and syntactical patterns together with possible source-specific structural variants and labels accompanying the relevant information. Although this by-hand approach requires a skillful user to design the extraction schema, we believe that in many cases it is faster and more reliable than using a learning approach to develop an individual wrapper for each source. Moreover, as shown in chapter 4, our algorithm can be used in a covering layer over existing site-specific wrappers that are used to apply their powerful features on selected fragments of the source.

As the support for semi-structured sources can be treated as an extension of the plain text approach, we present the basic ideas, extraction schema, and algorithm for extraction first from a plain text source (chapter 2). Then the extensions related to semi-structured sources are described in chapter 3. Integration of the algorithm into a Web extraction system is proposed in chapter 4 and the results of a performance evaluation are summarized in chapter 5. Chapter 6 compares our approach to related work, and chapter 7 concludes the paper.

2 Extraction from Plain Text

In this chapter, we present the basic algorithm that extracts structures according to a given extraction schema from a plain text source. Before the extraction starts, the source text is white space normalized (any sequence of white spaces is merged into one blank space) and tokenized (the text is broken into a sequence of tokens). The following items are considered to be separate tokens: any sequence of letters, any sequence of digits, and any non-letter or non-digit character.

2.1 Extraction Schema

To guide the extraction process, an *extraction schema* has to be defined. Schema definitions are based on context free grammars. An extraction schema is expressed by a set of grammar rules that define how to deduce *interpretations* of a given source:

(1) *Interpretation* ':= R:' *RegularExpression*
(2) *Interpretation* ':=' *CollectionType* '(' $I_1[C_1][P_1]$ ',' … $I_n[C_n][P_n]$ ')' ['D:' *maxTokenDistance*]
(3) *Interpretation* ':=' *CollectionType* '(' $I_1[C'_1]$ $[P_1]$ ['tokenRangeOverlap'] ',' …
$I_n[C'_n]$ $[P_n]$ ['tokenRangeOverlap'] ')'

where

– *Interpretation* is a unique name for the interpretation defined by the rule
– *RegularExpression* is an arbitrary regular expression
– $I_1, I_2, \ldots I_n$ are lower-level interpretation types

```
Motherboard       := Bag(FormFactor?,
          ExpansionSlots? tokenRangeOverlap)
FormFactor        := R:ATX|Micro ?ATX
ExpansionSlots    := Bag(AgpSlot?, Slot*)

AgpSlot           := Bag(AgpKeyword, AgpFeature*)D:2
AgpKeyword        := R:AGP
AgpFeature        := R:Pro|4x|2x

Slot              := Bag(SlotType, QuantityPack?)D:2
SlotType          := R:ISA|PCI|CNR
QuantityPack      := Sequence(Amount, Label?)D:0
Amount            := R:[1-9]
Label             := R:x
```

Fig. 2. Example extraction schema

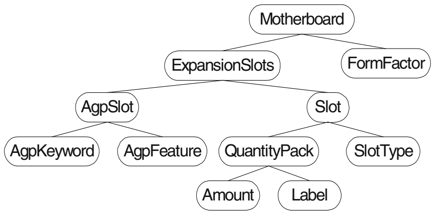


Fig. 3. Hierarchy of interpretation types

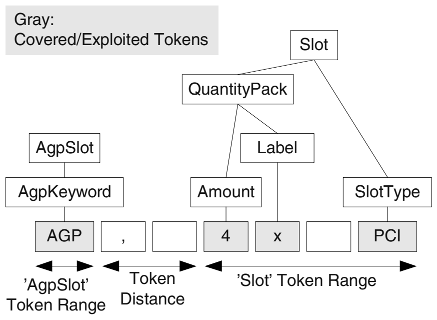


Fig. 4. Text tokenization and example interpretations

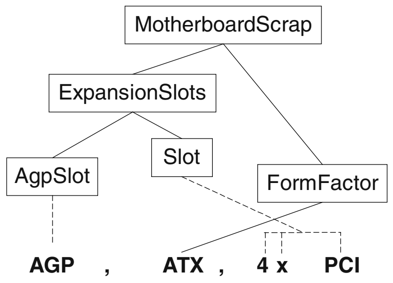


Fig. 5. Influence of the tokenRangeOverlap tag by ExpansionSlots

- *CollectionType* ∈ {'Sequence', 'Bag', 'Choice'} determines a type of the collection: ordered, unordered (allows arbitrary permutations on children), one of the children
- $C_1, C_2, \ldots C_n \in$ {'?', '*', '+'}, $C'_1, C'_2, \ldots C'_n \in$ {'?'} are optional occurrence. The meaning is as usual in the regular expression: '?' stands for optional occurrence, '+' for repetition, and '*' for optional repetition. When no occurrence constraint is defined, the interpretation is required to occur exactly once
- $P_1, P_2, \ldots P_n$ are integer relative preference values without direct influence on the structure. They are used to modify the standard mechanism for comparing the quality of interpretations (see chapter 2.2). The default value is 0.
- tokenRangeOverlap tag indicates the possible overlap of the token range by that of a sibling interpretation (see Figure 5)
- *maxTokenDistance* is the maximal allowed amount of source tokens between two adjacent lower-level interpretations. The default value is *unlimited*

Consider a source text "AGP, 4x PCI" and an extraction schema at Figure 2. The schema defines a hierarchy of interpretations. In our example the hierarchy is a tree (see Figure 3), however, any other hierarchical structure is allowed. Conceptually, most interpretations correspond to concepts of the application domain and reflect the concept hierarchy. At the lowest level of the hierarchy are so-called *atomic interpretations* that are assigned regular expressions (rule (1)). Whenever a regular expression is matched with a token or a sequence of tokens in the source, the corresponding atomic interpretation can be created and connected to the matched text (e.g. AgpKeword, Amount in Figure 4). Rules (2) and (3) define *structured interpretations* (e.g. AgpSlot, QuantityPack in Figure 4), which can be built as a collection of lower-level interpretations if they obey constraints imposed by the rules. It is a natural requirement that the atomic interpretations subsumed under a structured interpretation have to be connected to non-overlapping tokens of the source.

An important constraint in schema design is *maxTokenDistance* value. It specifies the maximal token distance between two adjacent interpretations in the collection. For example, the token distance between AgpSlot and Slot in Figure 4 is 2, thus, they could be joined to build ExpansionSlots interpretation.

During the extraction process, the extraction schema is used to deduce interpretations of a source. The qualitatively 'best' interpretation is returned as the result of the extraction.

2.2 Comparing Quality of Interpretations

The quality comparison allows choosing the qualitatively best interpretation from many valid interpretations of a single source. The most valuable property of an interpretation is the number of *exploited tokens*. These are all non white-space source tokens that were used to build the interpretation. In Figure 4, the number of exploited tokens (gray color) of Amount is 1, of QuantityPack is 2 and of Slot is 3. In order to have the finer means to compare two interpretations, we have developed the following heuristic-based sequence of comparison criteria:

1. Greatest number of exploited tokens.
2. Greatest height of the interpretation in the interpretation hierarchy, i.e. maximal distance of interpretation from any atomic interpretation. The higher interpretations are preferred.
3. Greatest weighted amount of exploited tokens in the interpretation. The weighting increases (by a factor of 1.1) the weight of an exploited token with each layer between the related atomic and root interpretations, i.e. an exploited token is more valuable when it is deeper in the interpretation tree. With a given relative preference value (optional parameter *P* in the rule), all atomic interpretations in the related sub-tree are considered, as if they are deeper by the parameter value.
4. Lowest weighted amount of all sub-interpretations in the interpretation tree. In contrast to the previous comparison, the weighting decreases (by a factor of 0.9) the weight of a sub-interpretation with each layer to the root interpretation. This criterion prefers (1) more economical (less total number of sub-interpretations) and (2) deeper determined interpretations.
5. Lowest width (token range) of the interpretation, i.e. lowest distance between the most left and most right exploited tokens. The narrower interpretations are better.
6. Lowest object id of the interpretation implementation object.

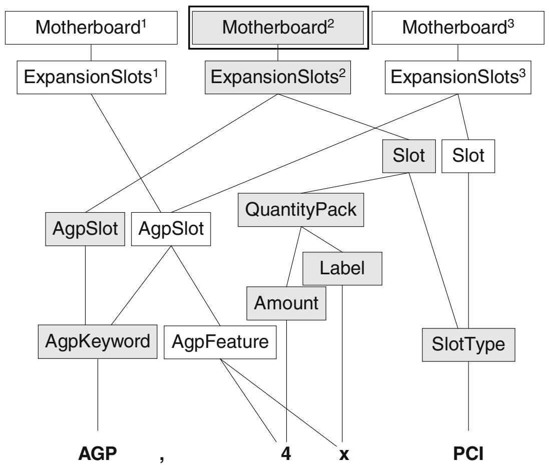


Fig. 6. Interpretations derived from the input text

Figure 6 shows a richer set of deduced interpretations for the extraction schema and source text from chapter 2.1. In this case, the Motherboard^2 is selected as the best of all interpretations (including the lower levels). The quality order of some other valuable interpretations is as follows:

Motherboard^1 $<_q$ ExpansionSlots^3 (Exploited tokens: $3 < 4$)
ExpansionSlots^3 $<_q$ ExpansionSlots^2 (Weighted amount of exploited tokens: $4.84 < 5.082$)
ExpansionSlots^2 $<_q$ Motherboard^3 (Maximal leaf distance of the roots: $4 < 5$)
Motherboard^3 $<_q$ Motherboard^2 (Weighted amount of exploited tokens: $5.324 < 5.5902$)

2.3 Extraction Algorithm

Given a source text, the algorithm searches through the space of possible interpretations for the qualitatively best one. Hereafter the term *partial interpretation* stands for an interpretation-like structure that contains only a subset of required child interpretations, and, thus, it cannot be recognized as a valid interpretation with respect to its definition.

The algorithm uses a bottom-up approach to derive higher-level interpretations. It runs in three phases:

1. The atomic, regular expression based interpretations are identified in the source and their ancestor interpretations are added until partial or top-level interpretations are reached.
2. Stepwise, going upwards in the interpretation hierarchy, new (partial-) interpretations are created as combinations of children of the existing interpretations.
3. The qualitatively best interpretation is returned as the result of extraction.

To explain the algorithm in more details, consider the example schema from chapter 2.1 and a source text “4x PCI”. Figure 7 shows notable steps of the extraction process. The state after the first phase is shown at Figure 7a. Partial interpretations, e.g. QuantityPack^2 that lacks a mandatory Amount child, are marked with the dotted line. On the contrary, one "PCI" token can be interpreted as a complete motherboard description.

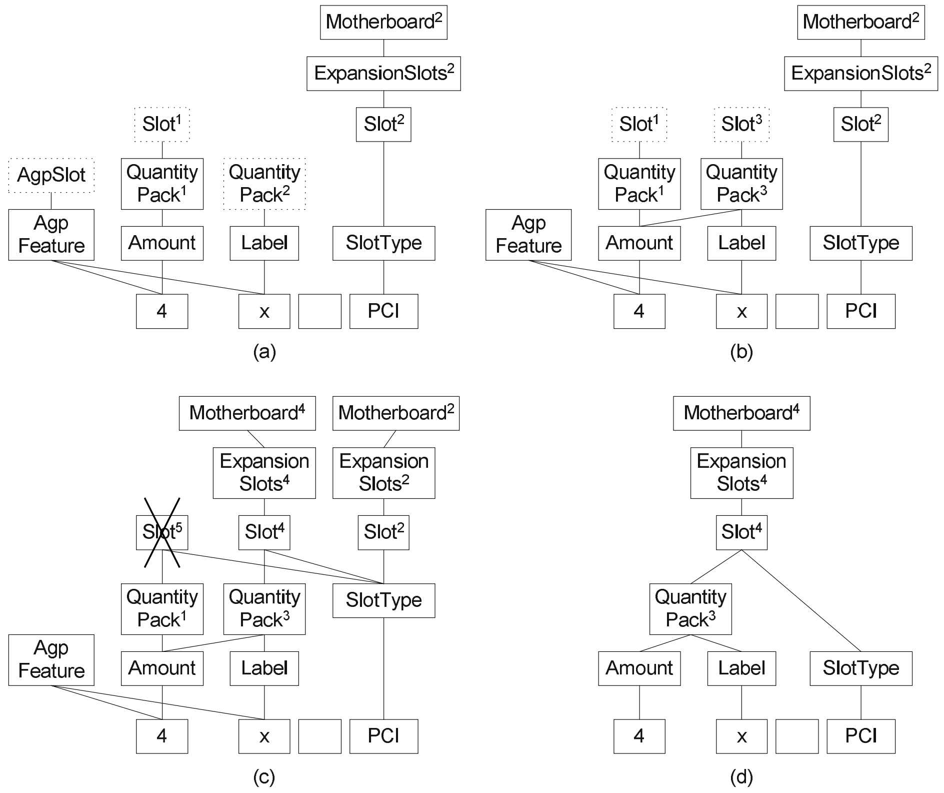


Fig. 7. Steps of deducing the best interpretation

In each step of the second phase, the algorithm is concerned with only one interpretation type. For each (partial-) interpretation of the given type in the right-to-left order, the algorithm repeatedly tries to combine its children with children of another 'more on the left lying' (partial-) interpretation of the same type in order to produce a new (partial-) interpretation. The new combination is not required to use all of the offered children. It utilizes only their qualitatively best selection. With each newly derived interpretation, its ancestors are added. Each new (partial-) interpretation can be later involved in this process and provide children for another combination. At the end of each step, all existing partial interpretations of the current type are removed, since they cannot be used either at any of higher levels or as a result of extraction.

In our example, the steps in the second phase are concerned with AgpSlot, QuantityPack, Slot, ExpansionSlots and Motherboard interpretations in the given order. Since there is no possibility to extend partial AgpSlot, it is removed at the end of the step. In the beginning of the next step, QuantityPack^2 is considered. Its child can be combined with the child of QuantityPack^1 into new QuantityPack^3 that can immediately generate ancestor Slot^3. Since QuantityPack^1 and QuantityPack^3 cannot be extended, the step ends with removing partial QuantityPack^2. The state after this step is shown at Figure 7b. The following step, which considers Slot interpretations, starts with combining Slot^2 and

Slot[3], which results in Slot[4] and its ancestors. Further, Slot[5] is produced as a result of combining Slot[2] and Slot[1]. However, it is recognized that Slot[5] is built on the same source text range as Slot[4]. Since any higher-level interpretation would prefer structurally equivalent and qualitatively better Slot[4] (more exploited tokens), Slot[5] is immediately discarded. Since Slot[3] and Slot[1] as well as all of the combination candidates in the following steps are overlapping, there are no more new combinations. This state is shown at Figure 7c that presents all interpretations from which the best one, Motherboard[4], is chosen as the result of extraction (see Figure 7d).

In order to decrease the complexity of the algorithm, we have implemented several methods to reduce the amount of generated and tested interpretations. All of the methods try either to avoid non-promising combinations or to remove an interpretation if a better one exists or is created within an overlapping text range. One of the latter methods was presented when discarding Slot[5] in the example above. Generally, the methods use monitoring token distances, structures, and so-called efficiency ranges which refer to remarkable quality of an interpretation inside a specific region of the source. Considering scalability in the length of the input source, the complexity of the extraction algorithm (considering maintenance of temporal structures) is $O(n^5 \log n)$ in the worst case, where n is the number of matched regular expressions in the text source. However, for real life applications we observed much better behavior. The algorithm works well and fast for typical sources with hundreds of tokens when the extraction schema contains up to circa hunderd interpretations of atomic types. The extraction task could become intractable for larger sources or schemas. One way to deal with the problem is to use a meticulous design of the extraction schema with as many restrictions as possible. Another way to speed up the extraction process is to convert the source into a semi-structured form and use the following adapted algorithm that exploits additional structure-based heuristics to decrease the amount of considered (partial-) interpretations.

3 Extraction from Semi-structured Sources

We consider any source in the form of a sorted tree, in which the leaves are text nodes and any non-leaf node contains a sorted set of leaf or non-leaf nodes, to be a semi-structured source. In the case of a Web page, we exploit only the tree structure and ignore the tag names. The basic idea of exploiting the tree structure is based on an observation on the context relationship of leaf tokens. It is supposed to be higher for tokens in one sub-tree than among tokens from different sub-trees. For example, any two data cells of a table in an html page are logically closer than the last cell in the table and the paragraph just below. Accordingly, there is a conclusion from the global perspective. Tokens prefer building relevant lower-level interpretations inside a sub-tree to building interpretations separately with tokens from sibling sub-trees. If we find several 'good' interpretations for a sub-tree, we can assume with high reliability that some of them are relevant parts of a relevant global interpretation. Consequently, we can discard all qualitatively inferior interpretations and consider only these 'good' ones when combining interpretations of sibling sub-trees at the higher level of the source structure. As discussed below, criteria for selecting the 'good' interpretations influence significantly the performance.

To support utilization of knowledge about source structure, we added a new rule to identify atomic interpretations:

(1b) *Interpretation* := 'L:' *regularExpression*

The regular expression is a pattern for a label that indicates the proximity of the text to be extracted by the interpretation. Generally speaking, we want to exploit all tokens in the nearest brother sub-tree to the sub-tree in which the match occurred (see Figure 8). The rule can be used advantageously to extract data from two-column tables or from sources that were specially prepared for this rule.

The adapted algorithm assigns each source node a set of interpretations of the sub-tree under the node. Before identifying atomic interpretations, the source is temporarily transformed into a pure text representation. The text nodes are normalized and tokenized, and then separated by line breaks. The regular expression matching, adapted to exploit rule (1b), is performed on this text representation. For each relevant sequence of tokens, the algorithm adds a new atomic interpretation to the set of the deepest source node that covers the sequence.

In the main phase, the algorithm traverses the source tree in the depth-first order. For each node, the algorithm adds 'good' interpretations of node's children to the node's set, and applies the second phase of the algorithm from the previous chapter to derive new (partial-) interpretations from the (partial-) interpretations in the set. Finally, the qualitatively best interpretation from the set assigned to the root node is returned as the result of extraction.

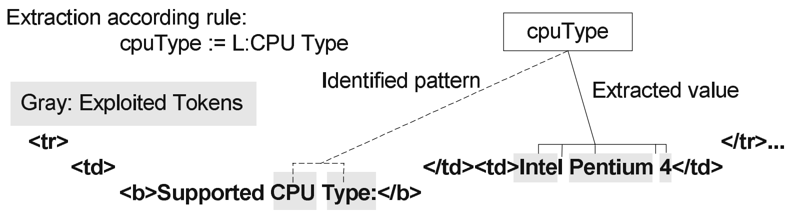


Fig. 8. Extraction using rule (1b)

Using the assumption on the closer contextual relationship of the tokens in one sub-tree, a suitable selection of 'good' interpretations can significantly speed up the extraction process, while maintaining quality. We tested several methods. The most remarkable results were the following:

a) No filtering at all – all derived interpretations were passed upwards. The performance and results were nearly the same as when extracted from the plain text representation of the source using the algorithm from the previous chapter. Minor differences appeared in the resulting structures when using loosely defined extraction schemas.
b) Use only (partial-) interpretations with the maximal amount of exploited tokens. The execution was approximately 10-100 times faster than in (a); however, the quality was in several cases significantly worse. Typically, rich partial interpretations pushed out all of the valid (and really relevant) ones with an inferior number of exploited tokens.

c) The 'good' set is created as a union of (1) all maximal (in sense of exploited tokens) valid interpretations, (2) all maximal valid interpretations that still can be extended, and (3) all partial interpretations that exploit more of or the same amount of tokens as the maximum in (1). The performance was only slightly worse than in (b). The quality of the result was typically the same as in (a), sometimes slightly worse. As this method seems to be most promising, we used it for performance evaluation in chapter 5.

4 Integration into a Web Extraction System

We have integrated the extraction algorithm on semi-structured sources into a system that extracts information in the form of XML documents from Web catalogs. The architecture of the system is shown in Figure 9. In the system, each Web source is represented by a URL pointing to its main catalog page. Each source-specific wrapper from the wrapper layer accesses the catalog pages linked from the main page and extracts the fragments containing the details of items of interest. The fragments are interpreted by the algorithm using a common extraction schema written in the form of an XML Schema (*x-schema*). The x-schema itself is an extended version of an XML Schema (a so-called *target schema*) describing XML structures to extract. While the target schema is a pure XML Schema without any dependence on any of the Web sources, the x-schema contains additional information: how to build target schema-like results from the fragments delivered from wrappers. The x-schema contains structural refinements and general as well as source-specific syntactical and structural regularities for recognizing relevant information. First, the algorithm deduces from a given fragment an interpretation according to the x-schema (can be expressed as an x-schema instance). Second, the interpretation is transformed into an XML instance of the target schema (a so-called *target result*) by removing extension tags introduced by the x-schema.

As our extraction approach was designed from the beginning to be used in XML-like environment, reconciliation between the grammar and above described XML-like approaches is quite natural. To use the algorithm, we must be able to transform an x-schema into an equivalent set of rules and formulate the extraction result in a relevant XML representation. The basic idea of the transformation is to map each defined XML element of a simple or complex type to an atomic respectively structured interpretation. In the XML-like result, each extracted value is returned as a string value of an element. Consequently, we support only those structural constructs of XML Schema definition language [10] that enable the transformation (e.g. elements, simple/complex type structures, XML collections, limited set of constraints). Other constructs are either ignored (attribute definitions) or not allowed (non-string data types, string data type without regular expression pattern, nested groups in the complex type definition…). Further, we have defined additional attributes of an external name space to be used in an x-schema. They are of two types: constraint attributes and visibility attributes. The former specifies the rule constraints (maxTokenDistance, tokenRangeOverlap), labels, relative preference values, and a simplified way to set up regular expression patterns. The visibility attributes are used to mask x-schema structural extensions from the target schema when producing a target result. While the XML Schema doesn't support the Bag collection type as defined in the grammar rule, an attribute

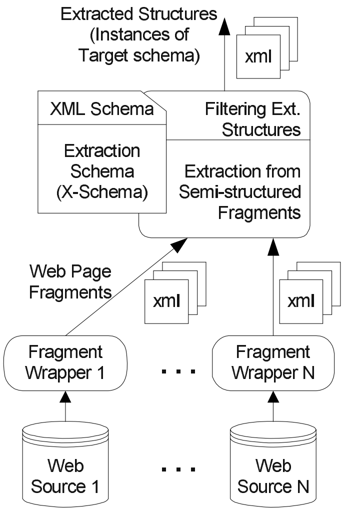

Extracted Structures (Instances of Target schema)
xml
XML Schema
Filtering Ext. Structures
Extraction Schema (X-Schema)
Extraction from Semi-structured Fragments
Web Page Fragments
xml
xml
Fragment Wrapper 1
. . .
Fragment Wrapper N
Web Source 1
. . .
Web Source N

Fig. 9. Architecture of the Web extraction system

changing the interpretation of the XML-sequence collection during the extraction is defined. Then the sub-elements are deduced unordered, and the ordering is performed during transformation of an interpretation into the XML format.

The process of building an extraction system for a given domain starts with specifying the target XML schema, i.e., the items of interest. Then the wrappers that deliver the fragments for extraction can be built for the sources. Stepwise for each source, the fragments are analyzed based on the regularities in the presentation of the relevant information. The observed patterns are used to extend the schema by refining the structures or adding constraint attributes.

The advantage of the proposed architecture is in the minimization of the effort to build the source-specific wrappers by postponing the main extraction task to the common level. Based on our experience, only several first analyzed sources, which provide information in the domain of interest, invoke significant changes in the x-schema. The others generally use a similar method in presenting information so that the invoked changes are minor (extending known labels by new keywords) or not needed.

In our implementation, we used simple XSLT-based wrappers and tested the approach on Web hardware catalogs (motherboards, graphic cards, CPU). In all cases, the average complexity of one source-specific wrapper was about 3 XSLT-style sheets of circa 2-5 templates each.

5 Performance Evaluation

We used the above architecture to test the extraction algorithm in the domain of motherboard descriptions. This domain offers non-trivially structured information as well as a relatively fixed and commonly adopted set of expressions to describe individual features. An initial version of the target XML schema was designed to accommodate general information about a motherboard with a logical structure as offered by most of the motherboard catalogs. Although the structure of the initial version was later slightly modified with respect to the capability of the extraction algorithm, the amount and character of desired information remained unchanged. The schematic description of the final target schema is shown at Figure 10. The stars indicate possible repetitive occurrence of the element; the italic labels introduce groups of non-value elements representing the listed items.

The extraction was performed on the web pages describing motherboard specifications from several randomly chosen motherboard producers (see Table 1). We tested the algorithm on semi-structured inputs as well as on their plain text representations. In the case of semi-structured input, the original page was transformed into XHTML format and used either directly (sources 5, 6, 2) or the relevant extraction fragment was obtained using one simple XPath expression (sources 1, 3, 4, 7).

The extraction quality was assessed through a comparison of extracted interpretations (instances of the target schema) to interpretations produced by a human interpreter (*correct interpretations*). The quality factor is expressed using precision and recall measures [1]. In our case, the recall is a ratio of the correctly extracted leaf elements to the leaf elements of the correct interpretation. The precision factor is a ratio of the correctly extracted leaf elements to all extracted leaf elements.

To extend the above target schema into the x-schema suitable for extraction, a training set of ten representative pages from each of the first two sources was used. During this process, the schema structure was refined and extended by accommodating keyword patterns that typically accompany the relevant information. Additionally, several typical synonyms of the observed keyword patterns were added. The x-schema was tuned to achieve 100% precision and recall on the training set. The final

- cpuSupport
    - producer (enumeration: AMD, Intel)
    - *Socket type*: Socket A, Socket 370, Socket 423, …
    - cpuInfo*
        - *Cpu type*: Pentium II/III/4, AMD K7,Athlon, …
        - relatedFrontSideBusFrequency (integer)
        - cpuMaxSpeed
            - frequency (decimal)
            - unit (enumeration: MHz, GHz)
    - cpuMaxSpeed (integer, or integer+ (for AMD processors))
- memorySupport
    - *Memory type*: DDR, RAM, SDRAM, DIMM, …
    - memorySlot*
        - pinCount (enumeration: 184, 168, 144)
        - slotAmount (enumeration: one, two,…; or int <= 10)
    - maxSupportedMemory
        - amount (decimal)
        - unit (enumeration: MB, GB)
- expansionSlots
    - agpSlot
        - feature (enumeration: 2x, 4x, 8x, Universal)
    - nonAgpSlot*
        - slotType (enumeration: PCI, ISA, CNR, USB, AMR)
        - slotAmount (enumeration: one, two,…; or int <=10)
- frontSideBusFrequency* (enumeration: 66, 100, 133, …, 533)
- formFactor (enumeration: ATX, Flex ATX, Micro ATX)

Fig. 10. Schematic performance evaluation schema

x-schema had approximately twice as many elements as the target schema. Subsequently, the extraction was performed on five randomly chosen pages from each of the seven selected sources (in the case of the first two sources only those pages that were not included in the training set were used). Table 1 shows the extraction results: recall and precision ranges, and details on the execution times.

Table 1. Results of performance evaluation

Source Nr.	Semi-structured source			Plain text source		
	Recall (%)	Precision (%)	Exec. Time (s)	Recall (%)	Precision (%)	Exec. Time (s)
1.	95.5 – 100	100	0.17 – 0.22	95.5 – 100	100	0.65 – 1.61
2.	95.0 – 100	100	0.11 – 0.16	100	100	0.18 – 0.60
3.	89.5 – 95.6	100	0.20 – 0.82	94.5 – 95.5	100	0.85 – 4.43
4.	85.5 – 91.3	100	0.14 – 0.18	85.5 – 91.3	100	0.33 – 0.47
5.	42.1 – 58.3	100	0.14 – 0.17	53.8 – 75.0	86.0 – 90.1	0.28 – 0.35
6.	58.9 – 92.3	81.7 – 100	0.11 – 0.35	66.4 – 92.3	84.2 – 96.0	0.79 – 2.19
7.	59.7 – 85.7	92.3 – 100	0.17 – 0.25	65.6 – 85.7	92.3 – 100	0.10 – 1.66

1. www.abit.com.tw	3. www.asus.com	5. www.epox.com	7. www.tyan.com
2. www.shuttleonline.com	4. www.biostar.com.tw	6. www.msi.com.tw	

The results demonstrate that the patterns observed from a few pages of one source can be reused with high reliability for other pages of the same source. Additionally, the keyword patterns were also good enough to cover successfully sources with a similar way of presenting information (3, 4). Interesting is the relatively high precision on sources 5 and 7, despite the low recall. The low precision in source 6 was caused by using the same keywords to express other information. The low recall for source 5 was caused by an atypical page layout in comparison to the other sources. The execution times document the difference in computational efficiency between the two versions of the algorithm.

After studying the above results, the x-schema was extended to improve extraction from sources 3 – 7 to achieve 100% precision and at least 90% recall on the tested pages. While we extended only the existing regular expressions with new keywords in sources 3, 4, and 7, we introduced several structural changes in sources 5 and 6.

6 Related Work

Although finite-automaton approaches [9, 6, 8] have an advantage that they provide typically a learning algorithm to automate wrapper generation, the generated wrappers are dependent on the presentation layer of a source, i.e., html tags. Consequently, they cannot be used for any new source without a training phase. Steps to simplify adaptation of a system to new sources have been done by Hong and Clark [5] who use a separate domain-specific knowledge coded in declarative rules and a learning approach exploiting stochastic context-free grammars. However, the generated wrappers still rely on html structuring. In contrast to that, our approach allows to utilize the domain-specific patterns without any dependence on the html code or structure.

Gao and Sterling [4] introduce a hybrid representation method for schemas, consisting of a concept hierarchy and a set of knowledge unit frames at the lowest level in the hierarchy. The knowledge unit frames represent patterns to identify and extract text data. The extraction process runs in two subtasks. The first task extracts the knowledge units. The second task groups the units into concepts, while the lower level boundaries of the semi-structured source are stepwise removed and higher-level concepts are derived. The approach provides a fast extraction algorithm; however, it lacks a more sophisticated mechanism to solve any ambiguity of a given source. The fixed assignment of the best knowledge unit for a text fragment is done during the first task, according to a certainty factor defined in the knowledge unit (sub-) frame description. According to our analysis that lead to our approach, one cannot determine in many cases the right meaning of a text fragment before considering its integration into higher-level concepts that exploit text fragments from different parts of the source structure. In addition, our approach is able to express more constraints on the concept relationship (e.g. maximal token distance, relative preference value).

Jedi [3] seems to be one of the outstanding representatives of approaches that use handcrafted grammars. Its powerful feature is the possibility to modify results of the parsing process by procedures assigned to grammar rules. On the other hand, they provide no means to modify the default method of dealing with ambiguous sources. The most left interpretation is selected without considering other features, which is an obstacle when processing highly irregular sources. Not least, the Jedi's grammars do not allow permutations of concepts.

7 Conclusion

In this paper we have presented a novel approach to extract catalog descriptions from heterogeneous Web sources that belong to the same product domain. As the basic assumption, the approach presumes existing common domain knowledge, namely, a widespread adopted basic terminology and at least loose logical similarity of the description structures. The observed general structural and syntactical patterns are accommodated in a common hand-crafted extraction schema together with possible structural and syntactical extensions introduced by individual Web sources. The extraction schema describes the structure of the information to extract and guides a specially designed algorithm that infers the best interpretation of a given semi-structured or plain text input. We have integrated the algorithm into a Web extraction system and tested it on real data. After a training phase, the system was able to deliver results of relatively high quality, even for sources that were not covered in the training phase.

The uniqueness of our system is in the expressiveness of the extraction schema supported by a specially designed algorithm, which allows high flexibility in data extraction from multiple sources. Currently, we are using the system as a data source in the XI^3 project. For the future, additional research to support design of the extraction schema is needed.

References

1. L. Eikvil. Information extraction from world wide web: a survey. Technical Report 945, Norwegian Computing Center, 1999
2. R. Gaizauskas, Y. Wilks: Information Extraction: Beyond Document Retrieval. Technical Report CS-97-10, Department of Computer Science, University of Sheffield, 1997
3. G. Huck, P. Fankhauser, K. Aberer, E. Neuhold: Jedi: Extracting and Synthesizing Information from the Web, In Proceedings of the 3rd IFCIS international Conference on Cooperative Information Systems, COOPIS' 98, New York, 1998
4. X. Gao, L. Sterling: Semi-Structured Data Extraction from Heterogeneous Sources. In David G. Schwartz, Monica Divitini, Terje Brasethvik(editors), Internet-based organisational memory and knowledge management, Part 2, Chapter 5, pages 83–102, The idea group Publisher, 1999
5. T. W. Hong and K. L. Clark: Using grammatical inference to automate information extraction from the web. In Principles of Data Mining and Knowledge Discovery, pages 216–227, 2001
6. C-H. Hsu and M-T. Dung: Generating Finite-State Transducers for Semistructured Data Extraction from the Web. Information systems, Vol. 23. No. 8, pp. 521–538, 1998
7. W. Kazakos, G. Nagypal, A. Schmidt, P. Tomczyk. XI^3 – Towards an Integration Web. In Proceedings of WITS02, Barcelona, December 2002
8. R. Kosala, J. Van den Bussche, M. Bruynooghe and H. Blockeel: Information Extraction in Structured Documents using Tree Automata Induction. In Principles of Data Mining and Knowledge Discovery, Proceedings of the 6th International Conference (PKDD-2002), pp. 299–310
9. N. Kushmerick, B. Thomas: Adaptive information extraction: Core technologies for information agents. In Intelligent Information Agents R&D in Europe: An AgentLink perspective. Springer-Verlag. In Press
10. W3C XML Schema http://www.w3c.org/XML/Schema.

UCYMICRA: Distributed Indexing of the Web Using Migrating Crawlers

Odysseas Papapetrou, Stavros Papastavrou, and George Samaras

Computer Science Department, University of Cyprus
75 Kallipoleos Str., P.O. Box 20537
{cs98po1,stavrosp,cssamara}@cs.ucy.ac.cy

Abstract. Due to the tremendous increase rate and the high change frequency of Web documents, maintaining an up-to-date index for searching purposes (search engines) is becoming a challenge. The traditional crawling methods are no longer able to catch up with the constantly updating and growing Web. Realizing the problem, in this paper we suggest an alternative distributed crawling method with the use of mobile agents. Our goal is a scalable crawling scheme that minimizes network utilization, keeps up with document changes, employs time realization, and is easily upgradeable.

1 Introduction

Indexing the Web has become a challenge due to the Web's growing and dynamic nature. A study released in late 2000 reveals that the static and publicly available Web (also mentioned as surface web) exceeds 2.5 billion documents while the deep Web (dynamically generated documents, intranet pages, web-connected databases etc) is almost three orders of magnitude larger [20]. Another study shows that the Web is growing and changing rapidly [17, 19], while no search engine succeeds coverage of more than 16% of the estimated Web size [19].

Web crawling (or traditional crawling) has been the dominant practice for Web indexing by popular search engines and research organizations since 1993, but despite the vast computational and network resources thrown into it, traditional crawling is no longer able to catch up with the dynamic Web. Consequently, popular search engines require up to 6 weeks in order to complete a crawling cycle for the non-paying sites [1]. Moreover, the centralized gathering nature of traditional crawlers and the lack of cooperation between major commercial search engines are two more reasons toward that.

The absence of a scalable crawling method triggered some significant research in the past few years. For example, Focused Crawling [6] was proposed as an alternative method but it did not introduce any architectural innovations since it relied on the same centralized practices of traditional crawling. As a first attempt to improve the centralized nature of traditional crawling, a number of distributed methods have been proposed (Harvest [4, 5], Grub [14]) and are discussed later on.

In this paper, we propose the UCYMicra, a distributed methodology for crawling the Web with the use of mobile agents. The driving force behind UCYMicra is the employment of mobile agents migrating from the search engine to the Web servers, and remaining there to crawl and monitor Web documents for updates. UCYMicra utilizes related concepts on Distributed Crawling introduced earlier in [4, 11, 12], to suggest a complete distributed crawling strategy aligned with the characteristics of mobile agents, aiming at (a) minimizing network utilization by avoiding the transmission of unnecessary data between Web servers and the search engine site, (b) keeping up with document changes by performing on-site monitoring that allows fast updates on the search engine database, (c) avoiding unnecessary overloading of the Web servers by employing time realization, and (d) being upgradeable at run time.

This introduction is followed by a description of the shortcomings caused by traditional crawling, and Section 3 describes related work. In Section 4, we present our crawling methodology and in Section 5 evaluate its performance. In Section 6, we discuss the advantages of our method, and we conclude in Section 7 with a reference to our ongoing work.

2 The Problem with the Traditional Crawling

For the last decade, single-sourced (or centralized) Web Crawling has been the driving force behind the most popular commercial search engines. The Google [3] and the AltaVista [15] search engines employ an incremental farm of local running crawlers in order to reach a daily dose of a few hundred millions downloads per day, while the allocation of more computational resources is often announced.

However, with the great expansion of the web, especially in the past four years, the traditional crawling model appears inadequate to adjust to the new facts since crawlers are no longer able to download documents with the daily rate required to maintain an updated index of the web. A relatively recent survey suggests that no search engine succeeds coverage of more than 16% of the estimated web size [19].

More specifically, the traditional crawling model fails for the following reasons:

- The task of processing the crawled data introduces a vast processing bottleneck at the search engine site. Distributed processing of data (at the remote Web servers), which would have alleviated the bottleneck, is not available through the current HTTP protocol used by crawlers. Current practices to alleviate this bottleneck are focused in the addition of more computational resources. However, the latter complicates resources coordination and increases the network traffic and cost.
- The attempt to download thousands of documents per second creates a network and a DNS lookup bottleneck. While the latter bottleneck can be partially removed using a local DNS cache, the former can only be alleviated by constantly adding more network resources.
- Documents are usually downloaded by the crawlers uncompressed increasing in this way the network bottleneck. In general, compression is not under full facilitation since it is independent from the crawling task and cannot be forced by the crawlers[1]. In addition, crawlers download the entire contents of a document, including useless information such as scripting code and comments, which are rarely necessary for the document processing.

[1] Document compression before transmission is not always available in Web servers. Web server administrators disable compression to save computational resources

- The vast size of the Web makes it impossible for crawlers to keep up with document changes. The re-visiting frequency for non-sponsored documents, in some of the biggest commercial search engines, varies from 4 to 6 weeks. As a result, search engines deliver old content in search queries.

Moreover, the current crawling model has negative consequences for the complete Internet infrastructure:

- The simultaneous execution of many commercial crawlers generates huge non-user related Internet traffic and tends to overload public Web servers, Domain Name servers, and the underlying Internet infrastructure (routers and networks).
- Not every crawler employs time realization. At peak time, web-servers may be "bombed" with HTTP GET requests generated by a Web crawler. Consequences may vary from a simple delay to a complete denial of service.

3 Related Work

In this Section, we discuss previous work done on achieving better Web coverage. First, we discuss Focused Crawling, a methodology where Web Crawlers limit their search on a particular topic. We then elaborate on Parallel and Distributed Crawling work.

3.1 Focused Crawling

Focused crawling [6] was introduced as an alternative to traditional crawling and aimed at (a) alleviating the scalability setback, and (b) improving search results. Driven by a specific topic (namely a set of keywords), focused crawling targets and monitors only a small fraction of the Web and therefore achieves high-quality indexing on a specific subject. More specifically, focused crawlers download and process only the links that their content is expected to be relevant to the topic of interest.

Significant research has been done in focused crawling with several suggestions for algorithms that filter the URLs and drive the crawlers [7, 10]. However, focused crawling suffers from the same scalability problems with traditional crawling since they both target the same expanding and constantly changing Web.

3.2 Parallel and Distributed Crawling

The concept of parallel and distributed crawling refers to multiple crawling units running at the same time. In parallel crawling [9], the crawling units run within the local environment of a single company, whereas in distributed crawling, the crawling units are geographically spread. Parallel and distributed crawling is a natural evolution toward accelerating crawling.

Mercator [15], an experimental parallel crawling architecture that was used later on in AltaVista's Search Engine 3, was attacking the scalability problem of crawling by adding more parallel crawlers on the local network. In addition, parallel crawling techniques have been used in the academic version of Google, while sufficient research has been done for more efficient task scheduling and distribution of the crawling task in [9].

Despite the falling prices in hardware and lower connectivity rates, however, parallel distribution of the crawling task within local resources fails to keep pace with the growing Web due to its centralized nature. Downloading of documents and DNS lookup share the same network resources. In addition, coordination of the crawling units and resource management generate a significant network overhead.

To resolve the problems of parallel crawling, non-local distribution of the crawling units was introduced. The first well-documented research dates in early 1994 with the Harvest Information Discovery and Access System [4, 5]. The Harvest prototype was running gatherers (brokers) at the information sources sites (namely the Web servers). Gatherers not only collected data through HTTP and FTP, but they also filtered and compressed data before transmitting it to a centralized database. Unfortunately, while similar software is still used for efficient in-house crawling, the Harvest project failed to become accepted due to lack of flexibility, and administrative concerns.

Grub [14], a recently launched project under the Open-source license, implements a distributed crawling methodology in a way similar to the SETI project [23]. Distributed crawlers collect and process data from local and remote sources and send information to the search engine for updates. However, the control of the whole architecture is centralized since a central server defines which URLs are to be crawled by which distributed crawlers.

J. Hammer and J. Fiedler in [11, 12] initiated the concept of Mobile Crawling by proposing the use of mobile agents as the crawling units. According to Mobile Crawling, mobile agents are dispatched to remote Web severs for local crawling and processing of Web documents. After crawling a specific Web server, they dispatch themselves either back at the search engine machine, or at the next Web server for further crawling. While the model offers speed and efficiency compared to current crawling techniques, important issues are to be resolved such as (a) a more efficient resource management (which is recognized from the writers as important future work), and (b) a more decentralized control of the architecture. In addition, this methodology requires from the mobile agents to constantly move through the Internet since there is no notion of the '*parking agent*'. The absence of a parked/stationed agent at the Web server site precludes an immediate way of promptly monitoring the Web server's documents for changes.

The Google Search Appliance [13], a recently launched commercial package by Google, offers local crawling and processing of a company's Web and document servers by plugging into the local network a Linux computer. This hardware/software solution, however, comes at a great cost and installation overhead. Furthermore, a closed Linux box is an approach that could not serve more that one search engine simultaneously, meaning that a similar package from another search engine would demand another similar box plugged in the local network.

4 The UCYMicra Crawling System

We introduce UCYMicra, a crawling system that utilizes concepts similar to those found in Mobile and distributed Crawling introduced in [11, 12, 4]. UCYMicra extends these concepts and introduces new ones in order to build a more efficient model for distributed Web crawling, capable of keeping up with Web document changes in real time. UCYMicra proposes a complete distributed crawling strategy by utilizing the mobile agents technology. The goals of UCYMicra are (a) to minimize network utilization (b) to keep up with document changes by performing on-site monitoring, (c) to avoid unnecessary overloading of the Web servers by employing time realization, and (d) to be upgradeable at run time.

The driving force behind UCYMicra is the utilization of mobile agents that migrate from the search engine to the Web servers, and remain there to crawl, process, and monitor Web documents for updates. Since UCYMicra requires that a specific mobile agents platform be running at the Web Server to be crawled, it is currently running under a distributed voluntary academic environment spanning in a number of continents. In the rest of this Section, we examine the architecture and functionality of UCYMicra.

4.1 Architecture of UCYMicra

UCYMicra consists of three subsystems, (a) the Coordinator subsystem, (b) the Mobile Agents subsystem, and (c) a public Search Engine that executes user queries on the database maintained by the Coordinator subsystem (the discussion of the public Search Engine is not in the scope of this paper).

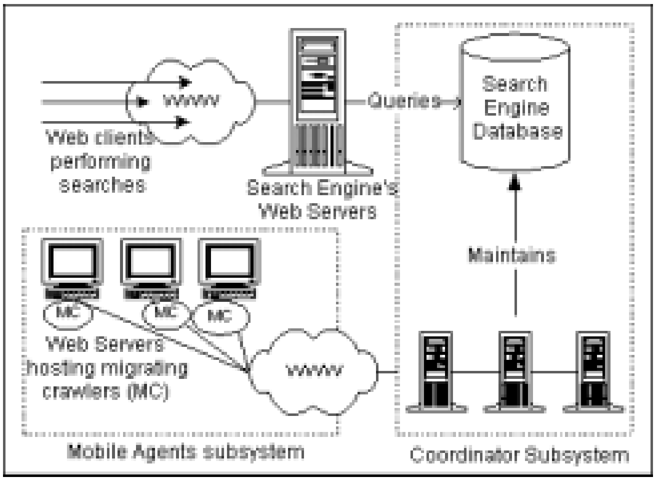

WWW
Web clients performing searches
Search Engine's Web Servers
Queries
Search Engine Database
Maintains
MC
MC
MC
Web Servers hosting migrating crawlers (MC)
WWW
Mobile Agents subsystem
Coordinator Subsystem

Fig. 1. Architecture of UCYMicra

The Coordinator subsystem resides at the Search Engine site and is responsible of maintaining the search database. In addition, it administers the Mobile Agents subsystem, which is responsible for crawling the Web. The Mobile Agents subsystem is divided into two categories of mobile agents namely the Migrating Crawlers (or Mobile Crawlers) and the Data Carries. The former are responsible for on-site crawling and monitoring of remote Web servers, and the latter for transferring the processed and compressed information from the Migrating Crawlers back to the Coordinator subsystem. Figure 1 illustrates the high-level architecture of UCYMicra.

4.2 The Coordinator Subsystem

Running locally to the search engine database, the Coordinator subsystem is primarily responsible of keeping it up-to-date by integrating fresh data received from the Mobile Agents subsystem. Second, but not less important, the Coordinator monitors the Mobile Agents subsystem in order to ensure its flawless operation. More specifically, the Coordinator subsystem:

1. Provides a publicly available Web-based interface through which Web server administrators can register their Web servers for participating in UCYMicra.
2. Creates and dispatches one Migrating Crawler for a newly registered Web server.
3. Monitors the lifecycle of the Migrating Crawlers in order to ensure their flawless execution.
4. Receives the Data Carriers in order to process their payload and integrate the result in the search engine database.

To avoid a potential processing bottleneck, the Coordinator is implemented to run distributed on several machines that collaborate with a simplified tuplespaces model (similar with Linda's distributed model described in [2]).

4.3 The Mobile Agents Subsystem

The Mobile Agents subsystem is the distributed crawling engine of our methodology and it is divided into two categories of Java mobile agents, (a) the Migrating Crawlers, and (b) the Data Carriers (the initial code of both agents resides at the Coordinator System where it is accessible for instantiation and dynamic installation).

In addition to its inherent mobile capabilities [8], a Migrating Crawler is capable of performing the following tasks at the remote site:

1. *Crawling*: A Migrating Crawler can perform a complete local crawling either though HTTP or using the file system in order to gather the entire contents of the Web server.
2. *Processing*: Crawled Web documents are stripped down into keywords, and keywords are ranked to locally create a keyword index of the Web server contents.
3. *Compression*: Using Java compression libraries, the index of the Web server contents is locally compressed to minimize transmission time between the Migrating Crawler and the Coordinator subsystem.
4. *Monitoring*: The Migrating Crawler can detect changes or additions in the Web server contents. Detected changes are processed, compressed and transmitted to the Coordinator subsystem.

A Data Carrier is a mobile agent dedicated in transferring processed and compressed data from a Migrating Crawler to the Coordinator subsystem for updating the search database. The choice of using mobile agents for data transmission over other network APIs (such as RMI, CORBA or sockets) is the utilization of their asynchronisity, flexibility and intelligence in order to ensure the faultless transmission of the data. For a descriptive analysis and comparison of the available Java network APIs, including mobile agents, please see [21].

4.4 Deployment of UCYMicra

Figure 2 illustrates how UCYMicra works. A registration procedure is required for a Web server to participate under the UCYMicra crawling system. The Coordinator subsystem provides the interface through which Web server administrators can define the registration parameters. Those parameters are divided into (a) the basic network information of the Web server to be crawled, and (b) the configuration options of the Migrating Crawler and the Data Carriers.

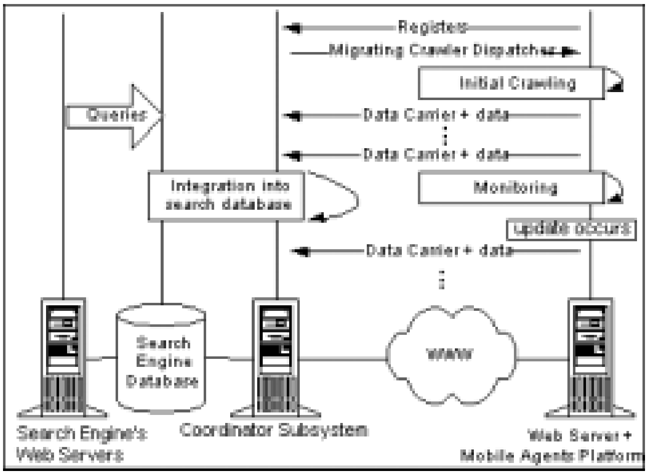


Fig. 2. UCYMicra at Work

The configuration options provide the Web server administrator with the ability to customize the behavior of the Migrating Crawler prior to its creation and relocation to the Web server site. More specifically, the Web server administrator defines[2]:

1. The time spans in which the Migrating Crawler is allowed to execute. Web server administrators might wish that the crawling and processing of data be executed during off-peak hours.
2. The sleep time between successive crawls used in monitoring the Web server contents for changes or additions. Web servers with frequent changes might consider having a smaller sleep time.

[2] The Web server administrator can also modify the configuration options at run time, in order to change the behavior of the Migrating Crawlers

3. The maximum packet size allowed of processed data for the Migrating Crawler to hold before dispatching it to the Coordinator subsystem for integration into the search database.
4. The maximum time-to-hold the processed data (packets) before dispatching it to the Coordinator subsystem. This parameter was introduced in order to avoid the case where processed data may become outdated before dispatching it to the Coordinator subsystem. Data may stall at the Migrating Crawler until the maximum packet size allowed is reached, in which case a data carrier is created, assigned the data, and dispatched.
5. Whether the Migrating Crawler will perform the crawling directly on the file system or through HTTP. For the former method, the absolute path(s) of the Web documents is provided. This method is recommended for Web servers with static Web content. For the second method, the Web server's URL(s) is provided and it is recommended for Web servers with dynamic content. Moreover, a combination of the above two can be used.

Following a successful registration of a Web server, the Coordinator subsystem creates a new instance of a Migrating Crawler and updates it with the registration options. The Migrating Crawler is then allowed to dispatch itself to the Web server.

Upon arrival at the Web server, the Migrating Crawler parks and suspends itself until the beginning of a pre-configured time span for execution arrives. For the first crawl, the Migrating Crawler will load, and process all the available Web documents at the Web server. Processing may include removal of useless html comments, scripting code, or even local extraction of keywords and ranking for each keyword (based on its occurrence frequency and other properties i.e. position on the document, font size and color). The processed data is being compressed and stored in memory until a data packet with the maximum allowed size can be assembled.

A Data Carrier agent is created for every packet assembled, or when the time-to-hold of a packet expires. Data carriers will then dispatch themselves to the Coordinator subsystem where they deliver their payload.

After the first crawl is completed and the Coordinator subsystem has updated the search database, the parked Migrating Crawler monitors the contents of the Web server for changes or additions of new documents. For documents retrieved through the file system, the last update time is used whereas for documents retrieved through HTTP, a conditional HTTP GET is used (using the If-Modified-Since header [16]). In either case, when the Migrating Crawler detects an update or an addition, it crawls and processes the affected documents, and transmits the new documents' index to the Coordinator subsystem. The transmission follows the rules defined in the pre-mentioned parameters (3) and (4), set by the Web server administrator.

4.5 Security in UCYMicra

Security is always an important aspect in every distributed system. In UCYMicra, security concerns are twofold:

- First, a secure runtime environment must be provided to Web server that will host the Migrating Crawlers. Such a runtime must guarantee that the migrating crawlers have access only to the necessary resources of their host, both software and hardware.

- Second, the runtime must also guarantee that the host machine has no access to the Migrating Crawler's processing code and data, in this way preventing malicious altering of the processing results.

Since the scope of this paper is to provide proof of concept for our crawling methodology, UCYMicra still does not employ security. Mobile Agents Security [24, 26], as well as the protection of the mobile agents themselves [18, 22] is a well-researched topic. It should be straightforward to embed security mechanisms in UCYMicra and this is part of our ongoing work.

5 Evaluating UCYMicra

5.1 Initial Experiments

We have performed an initial set of experiments in order to evaluate the performance of the UCYMicra crawling system against traditional crawling. We measure performance in terms of (a) *size of data transmitted* across the Internet, and (b) *total time required*, for the complete crawling of a given set of documents. For the time being, we study only this two obvious metrics and do not experiment with parameters such as document change frequency or update latency.

As opposed to the *size of data transmitted* metric that can be easily calculated in both approaches, the *total time required* metric was difficult to calculate, and depended of the experimental setup. This was mainly due to the following parameters that are beyond our control: (a) the network condition, under which our experiments were performed, and (b) the processing power and the load of the participating web servers during the experiments. We include, however, our time measurements in this paper to provide an additional insight on comparing the two approaches.

For the traditional crawling, the total time required metric stands for the time elapsed from the moment we feed the crawler's URL queue with the URLs to be crawled until all URLs have been successfully crawled and integrated into the search database. For UCYMicra, this is the time elapsed from the moment the Migrating Crawlers are dispatched from the Coordinator subsystem to the web servers that host the URLs to be crawled, until all the crawled contents of the URLs are carried by the Data Carriers back to the Coordinator subsystem and integrated into the search database.

Since it was not feasible to include commercial Web servers in our experiments, we have employed a set of ten Web servers within our voluntary distributed academic environment that span in several continents. Each Web server was hosting an average of 200 Web documents with an average document size of 25 Kbytes. The previous numbers yield 46.2 Mbytes of document data to be crawled by both the traditional crawling and the UCYMicra approach.

For the traditional crawling experiment, we have developed our own Java-based, multi-threaded crawling system and installed it on our own server machine. To trigger the crawler, we fed its URL queue with the addresses of the participating Web servers, and immediately the crawler started executing. Please note that in this experiment, the documents were downloaded uncompressed. In general, compression between a traditional crawler and a Web server is independent from the crawling task since a traditional crawler cannot enforce it.

For the UCYMicra evaluation, we have developed and installed the Coordinator Subsystem on our server machine. The Migrating Crawlers and Data Carriers were developed using the Voyager Platform [25] and their bytecode was installed on the server machine. Additionally, we have installed the Voyager Runtime on each available Web server machine to be crawled. For practical reasons, we did not require from an administrator to register every participating Web server. Instead, we automatically registered all the participating Web servers under reasonable registration parameters.

After the experiment was initiated, Migrating Crawlers were initialized and dispatched to every registered Web server to locally perform crawling, processing and compression of documents (one Crawler per Web server). Each Migrating Crawler initiated a number of Data Carriers that transferred the processed and compressed data back to the Coordinator Subsystem for integration within the search database.

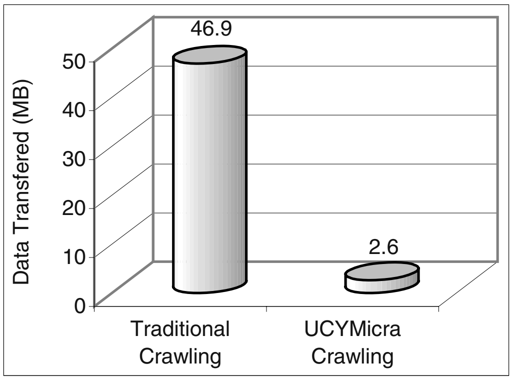


Fig. 3. Size of transferred data. UCYMicra outperforms traditional crawling by approximately generating 20 times less data

Figure 3 displays the size of transferred data by both the approaches in the experiment. Please note that in the 2.6 MB of data that UCYMicra generates, the code of the Migrating Crawlers and the Data Carriers that were dispatched to move the data is included, while in the traditional crawling, we include the data size of the HTTP requests and HTTP response headers. UCYMicra outperforms traditional crawling by generating 20 times less data. This is due to the following reasons:

- The Migrating Crawlers of UCYMicra crawl and process (with the same processing algorithm that the traditional crawler uses) the Web documents locally to the Web server. In this way, only the ranked keyword index of the Web server contents is transmitted to the Coordinator subsystem. In the traditional crawling approach, the crawler downloads the complete contents of a Web server.
- Before transmission to the Coordinator subsystem, the index of the Web server contents can always be compressed. A traditional crawler may request but cannot force a Web server to compress its contents before download. Unless the Web server has compression modules installed and enabled, the documents are delivered uncompressed.

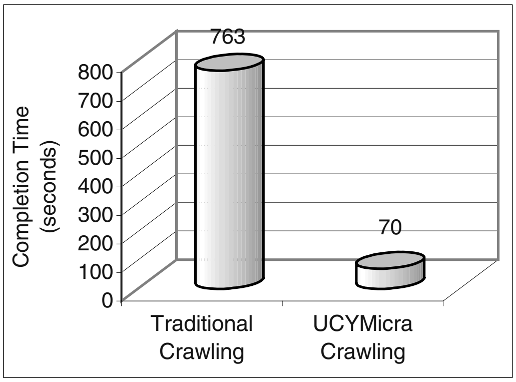


Fig. 4. Total time required for crawling. UCYMicra outperforms traditional crawling by an order of magnitude

Figure 4 displays the total time required by both the approaches to perform crawling. Similar to the previous results, UCYMicra requires one order of magnitude less time than the traditional approach to complete crawling. This is due to the following reasons:

- UCYMicra moves less data than the traditional approach and, therefore, requires less time.
- Processing of the Web documents is done in parallel by the Migrating Crawlers.

5.2 Analytical Experiments

To better understand the meaning of the results of our initial experiments, we performed an analytical second set of experiments by running one more variation of the traditional crawling approach, and three more variations of UCYMicra. The new traditional crawling variation downloads the Web documents compressed. As mentioned before, traditional crawlers cannot force the Web servers to compress documents, however, we were able to perform this experiment since we had control over the participating Web servers.

In the three new UCYMicra variations, the Migrating Crawlers differ in performing:

1. *Neither data processing nor compression.* In this variation, the Migrating Crawlers transmit to the Coordinator subsystem the entire contents of a Web server unprocessed (no keyword index generated) and uncompressed. This variation *emulates* traditional crawling by using the Data Carriers to move the data instead of the remote HTTP GET requests and HTTP response headers. In this case, none of the capabilities of the Migrating Crawlers is exploited (other than continuous monitoring).
2. *Data processing but not compression.* In this variation, the Migrating Crawlers transmit to the Coordinator subsystem the uncompressed keyword index.

3. *Compression but no data processing*. In this variation, the Migrating Crawlers transmit to the Coordinator subsystem the entire contents of a Web server compressed but not processed.

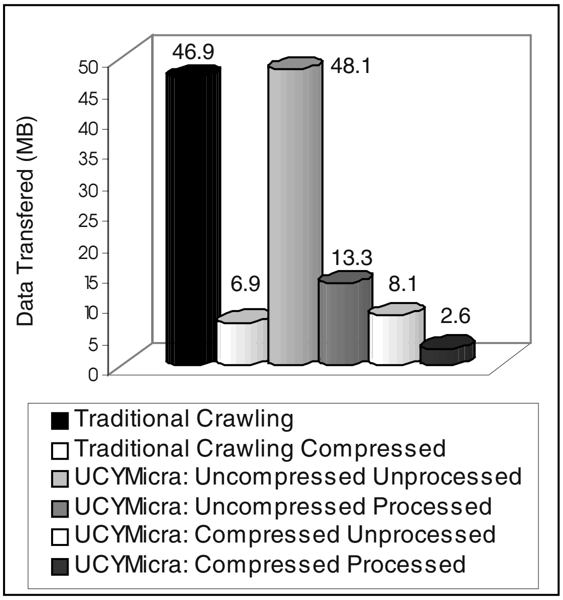


Fig. 5. Analytical performance results for size of transferred data

Figure 5 displays the analytical performance results for size of transferred data along with the initial performance results of Figure 3. Column 1 displays the initial results for the traditional crawling (with no compression). Column 6 displays the initial results for UCYMicra (with both processing and compression).

As seen in the results, it is clear that UCYMicra has to perform either processing (Column 4) or compression (Column 5) in order to outperform traditional crawling. However, both must be performed (Column 6: the original UCYMicra) in order to outperform the variation of the traditional crawling that downloads the Web documents compressed (Column 2).

The UCYMicra unprocessed-uncompressed variation (Column 3) that emulates traditional crawling is outperformed by traditional crawling (Column 1). This is because the execution code of all the Data Carriers that are dispatched to move the data is added to the total data size, which finally exceeds that of the traditional crawling.

Figure 6 displays the analytical performance results for the total time required for crawling.

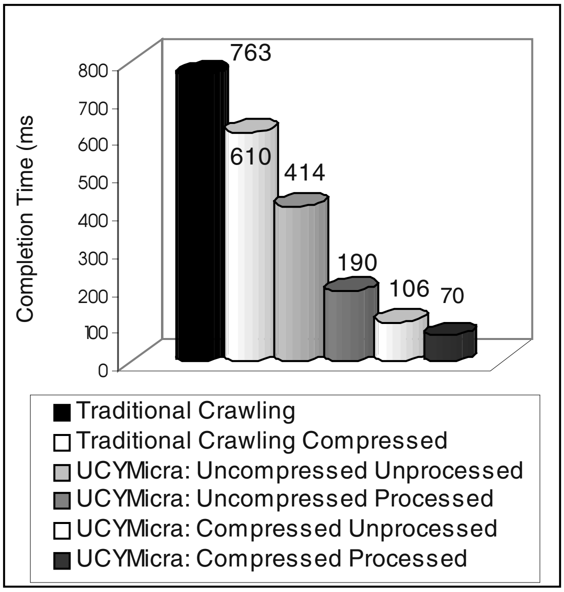


Fig. 6. Analytical performance results for total time required for crawling

In spite of the small size of transferred data that the traditional crawling with compression variation generates, our analytical experiments still prove its inefficiency in downloading Web documents with subsequent HTTP GET requests (Column 2). Even in the absence of remote compression and processing, UCYMicra outperforms the traditional crawling variation (with compression) since it transmits Web documents in a batched manner using the Data Carriers (Column 3).

6 Advantages of UCYMicra

As seen in the previous Section, UCYMicra outperforms traditional crawling without compression by a factor of ten in network utilization and total time required to perform crawling. Besides the performance gains, UCYMicra is a scalable distributed crawling architecture based on the mobile agents technology. More specifically, by delegating the crawling task to the mobile agent units, UCYMicra:

- Eliminates the enormous processing bottleneck from the search engine site. Traditional crawling requires that additional machines be plugged in to increase processing power, a method that comes at a great expense and cannot keep up with the current Web expansion rate and the document update frequency.
- Reduces the use of the Internet infrastructure and subsequently downgrades the network bottleneck by an order of magnitude. By crawling and processing Web documents at their source, we avoid transmission of data such as HTTP header information, useless HTML comments, and JavaScript code. In addition, data is

compressed before transmission, which is an option that cannot be forced by traditional crawlers.

- Keeps up with document changes. Migrating Crawlers monitor Web document for changes at their source and updates are immediately dispatched to the Coordinator subsystem. With traditional crawling, several weeks may pass before a Web site is revisited.
- Employs Time Realization. Migrating Crawlers do not operate on their hosts during peak time. As mentioned earlier, our approach requires from the administrator to define the permissible time spans for the crawlers to execute.
- Is upgradeable at real time. Newly developed crawling, processing, or compression code can be deployed over the entire UCYMicra since its crawling architecture is based on Java mobile agents.

7 Conclusions and Future Work

In this paper, we have presented a distributed crawling approach based on mobile agents. Our approach is streamlined not only in improving crawling performance, but also in handling Web document updates by performing onsite monitoring. UCYMicra presents a complete distributed crawling architecture that is efficient, flexible and quite scalable.

Our ongoing work focuses in extending UCYMicra to support a hybrid crawling mechanism that borrows technology from both the traditional and the fully distributed crawling system. Such a hybrid crawling system will support a hierarchical management structure that considers network locality, and will be able to perform traditional crawling in a completely distributed manner. Efficient algorithms for work delegation, administration, and result integration are currently under development.

References

1. Altavista Search Engine, Basic submit, Available at http://addurl.altavista.com/addurl/new
2. S. Ahuja, N. Carriero and D. Gelernter: Linda and Friends. IEEE Computer 19 (8), 1986, pp. 26–34.
3. S. Brin, and L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. In WWW7, Brisbaib, April 1998.
4. C. M. Brown, B. B. Danzig, D. Hardy, U. Manber, and M. F. Schwartz: The harvest information discovery and access system. In WWW2, Chicago, October 1994.
5. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz: Harvest: A Scalable, Customizable Discovery and Access System. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, August 1995.
6. S. Chakrabarti, M. van den Berg, B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. WWW8 / Computer Networks 31(11–16): 1623–1640 (1999).
7. S. Chakrabarti, K. Punera, and M. Subramanyam: Accelerated Focused Crawling through Online Relevance Feedback. In WWW2002, Hawaii, May 2002.
8. D. Chess, C. Harrison, and A. Kershenbaum: Mobile Agents: Are They A Good Idea? IBM research.

9. J. Cho, H. Garcia-Molina, Parallel Crawlers: In WWW2002, Hawaii, May 2002.
10. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori: Focused Crawling Using Context Graphs. VLDB 2000: 527–534.
11. J. Fiedler and J. Hammer: Using the Web Efficiently: Mobile Crawling. In Proc. of the Seventeenth Annual International Conference of the Association of Management (AoM/IAoM) on Computer Science, Maximilian Press Publishers, pp. 324–329. San Diego, CA, August 1999.
12. J. Fiedler and J. Hammer: Using Mobile Crawlers to Search the Web Efficiently. International Journal of Computer and Information Science, **1**:1, pp. 36–58, 2000.
13. Google Search Appliance. Available at http://www.google.com/services/.
14. Grub Distributed Internet Crawler. Available at www.grub.org.
15. A. Heydon, M. Najork: Mercator: A Scalable, Extensible Web Crawler. Compaq Systems Research Center. In WWW9, Amsterdam, May 2000.
16. Hypertext Transfer Protocol – HTTP/1.0, specification. Available at http://www.w3.org/.
17. B. Kahle: Achieving the Internet. Scientific American, 1996.
18. G. Karjoth, N. Asokan, C. Gulcu: Protecting the Computation Results of Free Roaming Agents. Second International Workshop on Mobile Agents, MA'98, LNCS-1477, 1998.
19. S. Lawrence, C. Lee Giles: Accessibility of information on the web. Nature, 400(6740):107–109, July 1999.
20. P. Lyman, H. Varian, J. Dunn, A. Strygin, and K. Swearingen: How much information? Available at http://www.sims.berkeley.edu/how-much-info.
21. M. Dikaiakos, G. Samaras: Quantitative Performance Analysis of Mobile Agent Systems. A Hierarchical Approach. Technical Report TR-00-2, Department of Computer Science, University of Cyprus, June 2000.
22. T. Sander and C. F. Tschudin: Towards Mobile Cryptography. In Proc. of the 1998 IEEE Symposium on Research in Security and Privacy, USA.
23. SETI: Search for Extraterrestrial Intelligence. Available at http://setiathome.ssl.berkeley.edu/.
24. V. Varadharajan. Security enhanced mobile agents: ACM Conference on Computer and Communications Security 2000: 200–209
24. Voyager Web site, by ObjectSpace. Available at http://www.recursionsw.com/products/voyager/voyager.asp.
26. N. Yoshioka, Y. Tahara, A. Ohsuga, S. Honiden: Security for Mobile Agents. AOSE 2000: 223–234.

Revisiting M-Tree Building Principles

Tomáš Skopal[1], Jaroslav Pokorný[2], Michal Krátký[1], and Václav Snášel[1]

[1] Department of Computer Science
VŠB–Technical University of Ostrava, Czech Republic
{tomas.skopal,michal.kratky,vaclav.snasel}@vsb.cz
[2] Department of Software Engineering
Charles University, Prague, Czech Republic
jaroslav.pokorny@ksi.ms.mff.cuni.cz

Abstract. The M-tree is a dynamic data structure designed to index metric datasets. In this paper we introduce two dynamic techniques of building the M-tree. The first one incorporates a multi-way object insertion while the second one exploits the generalized slim-down algorithm. Usage of these techniques or even combination of them significantly increases the querying performance of the M-tree. We also present comparative experimental results on large datasets showing that the new techniques outperform by far even the static bulk loading algorithm.

Keywords: M-tree, bulk loading, multi-way insertion, slim-down algorithm

1 Introduction

Multidimensional and spatial databases have become more and more important for different industries and research areas in the past decade. In the areas of CAD/CAM, geography, or conceptual information management, it is often to have applications involving spatial or multimedia data. Consequently, data management in such databases is still a hot topic of research. Efficient indexing and querying spatial databases is a key necessity to many interesting applications in information retrieval and related disciplines.

In general, the objects of our interests are spatial data objects. Spatial data objects can be points, lines, rectangles, polygons, surfaces, or even objects in higher dimensions. Spatial operations are defined according to the functionality of the spatial database to support efficient querying and data management. A spatial access method (SAM) organizes spatial data objects according to their position in space. As the structure of how the spatial data objects are organized can greatly affect performance of spatial databases, SAM is an essential part in spatial database systems (see e.g. [12] for a survey of various SAM).

So far, many SAM were developed. We usually distinguish them according to which type of space is a particular SAM related. One class of SAM is based on vector spaces, the second one uses metric spaces. For example, well-known data structures like kd-tree [2], quad-tree [11], and R-tree [8], or more recent ones like UB-tree [1], X-tree [3], etc. are based on a form of vector space. Methods for indexing metric spaces include e.g. metric tree [14], vp-tree [15], mvp-tree [5], Slim-tree [13], and the M-tree [7].

Searching for objects in multimedia databases is based on the concept of similarity search. In many disciplines, similarity is modelled using a distance function. If the well-known triangular inequality is fulfilled by this function, we obtain metric spaces. Authors of [9] remind that if the elements of the metric space are tuples of real numbers then we get a finite dimensional vector space.

For spatial and multimedia databases there are three interesting types of queries in metric spaces: range queries, nearest neighbours queries, and k-nearest neighbours queries. A performance of these queries differs in vector and metric spaces. For example, the existing vector space techniques are very sensitive to the space dimensionality. Closest point search algorithms have an exponential dependency on the dimensionality of the space (this is called the curse of dimensionality, see [4] or [16]).

On the other hand, metric space techniques seem to be more attractive for a large class of applications of spatial and multimedia databases due to their advantages in querying possibilities. In the paper, we focus particularly on improvement of the dynamic data structure M-tree. The reason for M-tree lies in the fact that, except Slim-trees, it is still the only persistent metric index. In existing approaches to M-tree algorithms there is a static bulk loading algorithm with a small construction complexity. Unfortunately, a querying performance of above-mentioned types of queries is not too high on such tree.

We introduce two dynamic techniques of building the M-tree. The first one incorporates a multi-way object insertion while the second one exploits the generalized slim-down algorithm. Usage of these techniques or even combination of them significantly increases the querying performance of the M-tree. We also present comparative experimental results on large datasets showing that the new techniques outperform by far even the static bulk loading algorithm. By the way, the experiments have shown that the querying performance of the improved M-tree has grown by more than 300%.

In Section 2 we introduce shortly general concepts of the M-tree, discuss the quality of the M-tree structure, and introduce the multi-way insertion method. In Section 3 we repeat the slim-down algorithm and we also introduce here a generalization of this algorithm. Experimental results and their discussion are presented in Section 4. Section 5 concludes the results.

2 General Concepts of the M-Tree

M-tree, introduced in [7] and elaborated in [10], is a dynamic data structure for indexing objects of metric datasets. The structure of M-tree was primarily designed for multimedia databases to natively support the similarity queries.

Let us have a metric space $\mathcal{M} = (D, d)$ where D is a domain of feature objects and d is a function measuring distance between two feature objects. A feature object $O_i \in D$ is a sequence of features extracted from the original database object. The function d must be a metric, i.e. d must satisfy the following metric axioms:

$$\begin{array}{rlr} d(O_i, O_i) = 0 & & \text{reflexivity} \\ d(O_i, O_j) > 0 & (O_i \neq O_j) & \text{positivity} \\ d(O_i, O_j) = d(O_j, O_i) & & \text{symmetry} \\ d(O_i, O_j) + d(O_j, O_k) \geq d(O_i, O_k) & & \text{triangular inequality} \end{array}$$

The M-tree is based on a hierarchical organization of feature objects according to a given metric d. Like other dynamic and persistent trees, the M-tree structure is a balanced hierarchy of nodes. As usually, the nodes have a fixed capacity and a utilization threshold. Within the M-tree hierarchy, the objects are clustered into metric regions. The leaf nodes contain entries of objects themselves (here called the ground objects) while entries representing the metric regions are stored in the inner nodes (the objects here are called the routing objects). For a *ground object* O_i, the entry in a leaf has a format:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, P(O_i))]$$

where $O_i \in D$ is the feature object, $oid(O_i)$ is an identifier of the original DB object (stored externally), and $d(O_i, P(O_i))$ is a precomputed distance between O_i and its parent routing object.

For a *routing object* O_j, the entry in an inner node has a format:

$$rout(O_j) = [O_j, ptr(T(O_j)), r(O_j), d(O_j, P(O_j))]$$

where $O_j \in D$ is the feature object, $ptr(T(O_j))$ is pointer to a covering subtree, $r(O_j)$ is a covering radius, and $d(O_j, P(O_j))$ is a precomputed distance between O_j and its parent routing object (this value is zero for the routing objects stored in the root). The entry of a routing object determines a metric region in space $\mathcal{M}$ where the object O_j is a center of that region and $r(O_j)$ is a radius bounding the region. The precomputed value $d(O_j, P(O_j))$ is redundant and serves for optimizing the algorithms upon the M-tree. In Figure 1, a metric region and

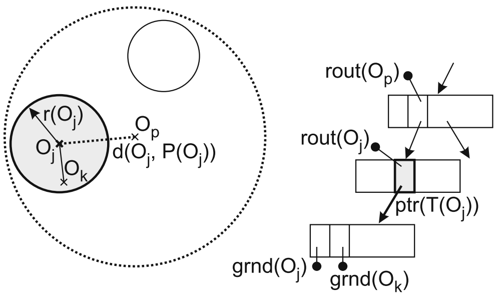


Fig. 1. A metric region and its routing object in the M-tree structure.

its appropriate entry $rout(O_j)$ in the M-tree is presented. For the hierarchy of metric regions (routing objects $rout(O)$ respectively) in the M-tree, only one invariant must be satisfied. The invariant can be formulated as follows:

• All the ground objects stored in the leafs of the covering subtree of $rout(O_j)$ must be spatially located inside the region defined by $rout(O_j)$. •

Formally, having a $rout(O_j)$ then $\forall O \in T(O_j), d(O, O_j) \leq r(O_j)$. If we realize, this invariant is very weak since there can be constructed many M-trees of the same object content but of different structure. The most important consequence is that many regions on the same M-tree level may overlap. An example

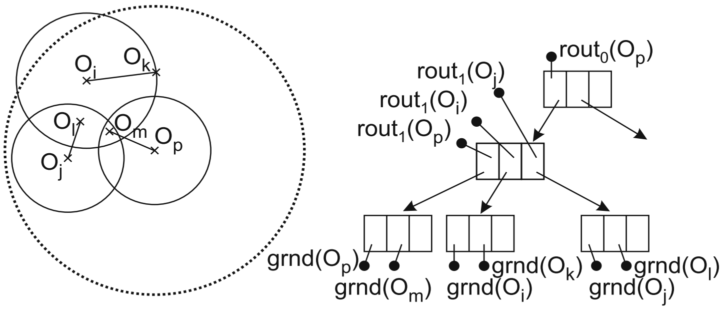


Fig. 2. Hierarchy of metric regions and the appropriate M-tree.

in Figure 2 shows several objects partitioned into metric regions and the appropriate M-tree. We can see that the regions defined by $rout_1(O_p)$, $rout_1(O_i)$, $rout_1(O_j)$ overlap. Moreover, object O_l is located inside the regions of $rout(O_i)$ and $rout(O_j)$ but it is stored just in the subtree of $rout_1(O_j)$. Similarly, the object O_m is located even in three regions but it is stored just in the subtree of $rout_1(O_p)$.

2.1 Similarity Queries

The structure of M-tree natively supports similarity queries. A similarity measure is here represented by the metric function d. Given a query object O_q, a similarity query returns (in general) objects close to O_q. The similarity queries are of two basic kinds: a *range query* and a *k-nearest neighbour query.*

Range Queries. A range query is specified as a *query region* given by a query object O_q and a query radius $r(O_q)$. The purpose of a range query is to return all the objects O satisfying $d(O_q, O) \leq r(O_q)$. A query with $r(O_q) = 0$ is called a *point query.*

***k*-Nearest Neighbours Queries.** A k-nearest neighbours query (k-NN query) is specified by a query object O_q and a number k. A k-NN query returns the first k nearest objects to O_q. Technically, the k-NN query can be implemented using the range query with a dynamic query radius. In practice, the k-NN query is used more often than the range query since the size of the k-NN query result is known in advance.

By the processing of a range query (k-NN query respectively), the M-tree hierarchy is being passed down. Only if a routing object $rout(O_j)$ (its metric region respectively) intersects the query region, the covering subtree of $rout(O_j)$ is relevant to the query and thus further processed.

2.2 Quality of the M-Tree

As of many other indexing structures, the main purpose of the M-tree is its ability to efficiently process the queries. In other words, when processing a similarity query, a minimum of disk accesses as well as computations of d should be performed. The need of minimizing the disk access costs[1] (DAC) is a requirement well-known from other index structures (B-trees, R-trees, etc.). Minimization of the computation costs (CC), i.e. the number of the d function executions, is also desirable since the function d can be very complex and its execution can be computationally expensive. In the M-tree algorithms, the DAC and CC are highly correlated, hence in the following we will talk just about "costs".

The key problem of the M-tree's efficiency resides in a quantity of overlaps between the metric regions defined by the routing objects. If we realize, the query processing must examine all the nodes the parent routing objects of which intersect the query region. If the query region lies (even partially) in an overlap of two or more regions, all the appropriate nodes must be examined and thus the costs grow.

In generic metric spaces, we cannot quantify the volume of two regions overlap and we even cannot compute the volume of a whole metric region. Thus we cannot measure the goodness of an M-tree as a sum of overlap volumes. In [13], a *fat-factor* was introduced as a way to classify the goodness of the Slim-tree, but we can adopt it for the M-tree as well. The fat-factor is tightly related to the M-tree's query efficiency since it informs about the number of objects in overlaps using a sequence of point queries.

For the fat-factor computation, a point query for each ground object in the M-tree is performed. Let h be the height of an M-tree T, n be the number of ground objects in T, m be the number of nodes, and I_c be the total DAC of all the n point queries. Then,

$$fat(T) = \frac{I_c - h \cdot n}{n} \cdot \frac{1}{(m-h)}$$

is the fat-factor of T, a number from interval $\langle 0, 1 \rangle$. For an ideal tree, the $fat(T)$ is zero. On the other side, for the worst possible M-tree the $fat(T)$ is equal to one. For an M-tree with $fat(T) = 0$, every performed point query costs h disk

[1] considering all logical disk accesses, i.e. disk cache is not taken into account

accesses while for an M-tree with $fat(T) = 1$, every performed point query costs m disk accesses, i.e. the whole M-tree structure must be passed.

2.3 Building the M-Tree

By revisiting the M-tree building principles, our objective was to propose an M-tree construction technique keeping the fat-factor minimal even if the building efforts would increase.

First, we will discuss the dynamic insertion of a single object. The insertion of an object into the M-tree has two general steps:

1. Find the "most suitable" leaf node where the object O will be inserted as a ground object. Insert the object into that node.
2. If the node overflows, split the node (partition its content among two new nodes), create two new routing objects and promote them into the parent node. If now the parent node overflows, repeat step 2 for the parent node. If a root is split the M-tree grows by one level.

Single-Way Insertion. In the original approach presented in [7], the basic motivation used to find the "most suitable" leaf node is to follow a path in the M-tree which would avoid any enlargement of the covering radius, i.e. at each level of the tree, a covering subtree of $rout(O_j)$ is chosen, for which $d(O_j, O) \leq r(O_j)$. If multiple paths with this property exist, the one for which object O is closest to the routing object $rout(O_j)$ is chosen.

If no routing object for which $d(O_j, O) \leq r(O_j)$ exists, an enlargement of a covering radius is necessary. In this case, the choice is to minimize the increase of the covering radius. This choice is thightly related to the heuristic criterion that suggests to minimize the overall "volume" covered by routing objects in the current node.

The single-way leaf choice will access only h nodes, one node on each level, as depicted in Figure 3a.

Multi-way Insertion. The single-way heuristic was designed to keep the building costs as low as possible and simultaneously to choose a leaf node for which the insertion of the object O will not increase the overall "volume". However, this heuristic behaves very locally (only one path in the M-tree is examined) and thus the most suitable leaf may be not chosen.

In our approach, the priority was to choose the most suitable leaf node at all. In principle, a point query defined by the inserted object O is performed. For all the relevant leafs (their routing objects $rout(O_j)$ respectively) visited during the point query, the distances $d(O_j, O)$ are computed and the leaf for which the distance is minimal is chosen. If no such leaf is found, i.e. no region containing the O exists, the single-way insertion is performed.

This heuristic behaves more globally since multiple paths in the M-tree are examined. In fact, all the leafs the regions of which spatially contain the object O are examined. Naturally, the multi-way leaf choice will access more nodes than h as depicted in Figure 3b.

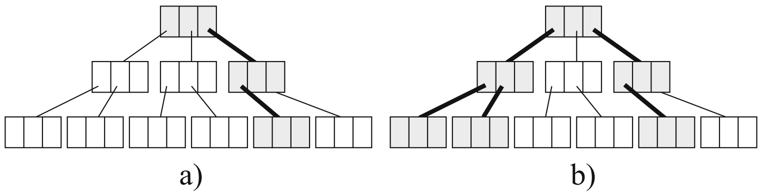


Fig. 3. a) Single path of the M-tree is passed during the single-way insertion. b) Multiple leafs are examined during the multi-way insertion.

Node Splitting. When a node overflows it must be split. According to keep the minimal overlap, a suitable splitting policy must be applied. Splitting policy determines how to split a given node, i.e. which objects to choose as the new routing objects and how to partition the objects into the new nodes.

As the experiments in [10] have shown, the `minMAX_RAD` method of choosing the routing objects causes the best querying performance of the M-tree. The `minMAX_RAD` method examines all of the $\frac{n(n-1)}{2}$ pairs of objects candidating to the two new routing objects. For every such a pair, the remaining objects in the node are partitioned according to the objects of the pair. For the two candidate routing objects a maximal radius is determined. Finally, such a pair $(rout(O_i), rout(O_j))$ for which is the maximal radius (the greater of the two radii $r(O_i), r(O_j)$) minimal is chosen as the two new routing objects.

For the object partition, a distribution according to *general hyperplane* is used as the beneficial method. An object is simply assigned to the routing object that is closer. For preservation of the minimal node utilization a fixed amount of objects is distributed according to the balanced distribution.

2.4 Bulk Loading the M-Tree

In [6] a static algorithm of the M-tree construction was proposed. On a given dataset a hierarchy is built resulting into a complete M-tree.

The basic bulk loading algorithm can be described as follows: Given the set of objects $\mathcal{S}$ of a dataset, we first perform an initial clustering by producing k sets of objects $\mathcal{F}_1, \ldots, \mathcal{F}_k$. The k-way clustering is achieved by sampling k objects $O_{f_1}, \ldots, O_{f_k}$ from the $\mathcal{S}$ set, inserting them in the sample set $\mathcal{F}$, and then assigning each object in $\mathcal{S}$ to its nearest sample, thus computing $k \cdot n$ distance matrix. In this way, we obtain k sets of relatively “close” objects. Now, we invoke the bulk loading algorithm recursively on each of these k sets, obtaining k sub-trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$. Then, we have to invoke the bulk loading algorithm one more time on the set $\mathcal{F}$, obtaining a super-tree $\mathcal{T}_{sup}$. Finally, we append each sub-tree $\mathcal{T}_i$ to the leaf of $\mathcal{T}_{sup}$ corresponding to the sample object O_{f_i}, and obtain the final tree $\mathcal{T}$.

The algorithm, as presented, would produce a non-balanced tree. To resolve this problem we use two different techniques:

– Reassign the objects in underfull sets $\mathcal{F}_i$ to other sets and delete corresponding sample object from $\mathcal{F}$.
– Split the taller sub-trees, obtaining a shorter sub-trees. The roots of the sub-trees will be inserted in the sample set $\mathcal{F}$, replacing the original sample object.

A more precise description of the bulk loading algorithm can be found in [6] or [10].

3 The Slim-Down Algorithm

Presented construction mechanisms incorporate decision moments that regard only a partial knowledge about the data distribution. By the dynamic insertion, the M-tree hierarchy is constructed in a moment when the nodes are about to split. However, splitting a node is only a local redistribution of objects. From this point of view, the dynamic insertion of the whole dataset will raise a sequence of node splits – local redistributions – which may lead to a hierarchy that is not ideal.

On the other side, the bulk loading algorithm works statically with the whole dataset, but it also works locally – according to a randomly chosen sample of objects.

In our approach we wanted to utilize a global mechanism of (re)building the M-tree. In [13] a post-construction method was proposed for the Slim-tree, called as *slim-down* algorithm. The slim-down algorithm was used for an improvement of a Slim-tree already built by dynamic insertions. The basic idea of the slim-down algorithm was an assumption that a more suitable leaf exists for a ground object stored in a leaf. The task was to examine the most distant objects (from the routing object) in the leaf and try to find a better leaf. If such a leaf existed the object was inserted to the new leaf (without the need of its covering radius enlargement) and deleted from the old leaf together with a decrease of its covering radius. This algorithm was repeatedly applied for all the ground objects as long as the object movements occured.

However, the experiments have shown that the original (and also cheaper) version of the slim-down algorithm presented in [13] improves the querying performace of the Slim-tree only by 35%.

3.1 Generalized Slim-Down Algorithm

We have generalized the slim-down algorithm and applied it for the M-tree as follows:

The algorithm separately traverses each level of the M-tree, starting on the leaf level. For each node N on a given level, a better location for each of the objects in the node N is tried to find. For a ground object O in a leaf N, a set of relevant leafs is retrieved, similarly like the point query does it by the multi-way insertion. For a routing object O in a node N, a set of relevant nodes (on the appropriate level) is retrieved. This is achieved by a modified range query, where the query radius is $r(O)$ and only such nodes are processed the routing objects of which *entirely* contain $rout(O)$. From the relevant retrieved nodes a node is chosen the parent routing object $rout(O_i)$ of which is closest to the object O. If the object O is closer to $rout(O_i)$ more than to the routing object of N (i.e. $d(O, rout(O_i)) < d(O, rout(N))$, the object O is moved from N to the new node. If O was the most distant object in N, the covering radius of its routing object $rout(N)$ is decreased. Processing of a given level is repeated as long as any object movements are occuring. When a level is finished the algorithm for the next higher level starts.

The slim-down algorithm reduces the fat-factor of the M-tree via decreasing the covering radii of routing objects. The number of nodes on each M-tree level is preserved since only redistribution of objects on the same level is performed during the algorithm and no node overflows or underflows (and thus node splitting or merging) by the object movements are allowed.

Example (generalized slim-down algorithm):
Figure 4 shows an M-tree before and after the slim-down algorithm application.

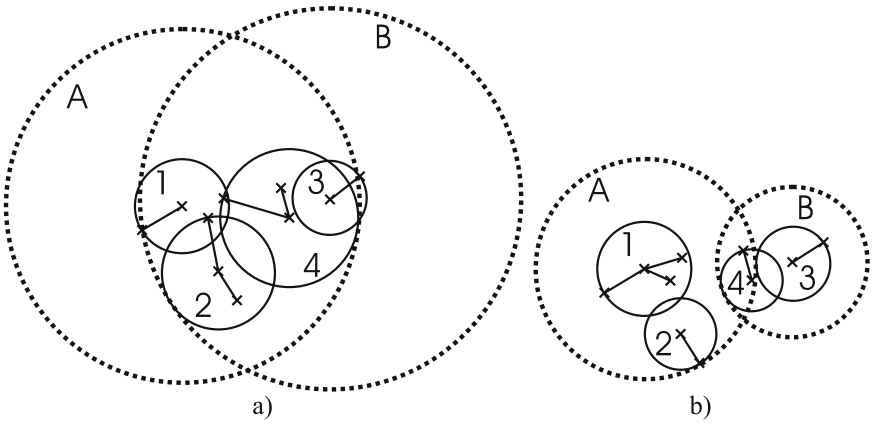


Fig. 4. a) M-tree before slimming down. b) M-tree after slimming down.

Routing objects stored in the root of the M-tree are denoted as A, B while the routing objects stored in the nodes of first level are denoted as 1, 2, 3, 4. In the leafs are stored the ground objects (denoted as crosses). Before slimming down, the subtree of A contains 1 and 4 while the subtree of B contains 3 and 2. After slimming down the leaf level, one object was moved from 2 to 1 and one object was moved from 4 to 1. Covering radii of 2 and 4 were decreased. After slimming down the first level, 4 was moved from A to B, and 2 was moved from B to A. Covering radii of A and B were decreased.

4 Experimental Results

We have completely reimplemented the M-tree in C++, i.e. we have not used the original GiST implementation (our implementation is stable and about 15-times faster than the original one). The experiments ran on an Intel Pentium®4 2.5GHz, 512MB DDR333, under Windows XP.

The experiments were performed on synthetic vector datasets of clustered multidimensional tuples. The datasets were of variable dimensionality, from 2 to 50. The size of dataset was increasing with the dimensionality, from 20,000 2D tuples to 1 million 50D tuples. The integer coordinates of the tuples were ranged from 0 to 1,000,000.

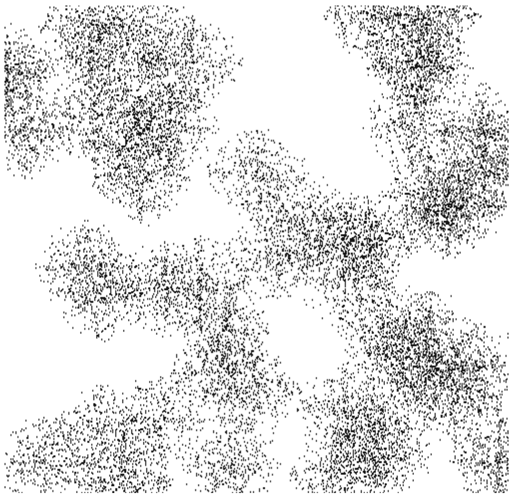


Fig. 5. Two-dimensional dataset distribution.

The data were randomly distributed inside hyper-spherical (L_2) clusters (the number of clusters was increasing with the increasing dimensionality – 50 to 1,000 clusters) with radii increasing from 100,000 (10% of the domain extent) for 2D tuples to 800,000 (80% of the domain extent) for 50D tuples. In such distributed datasets, the hyper-spherical clusters were highly overlapping due to their quantity and large radii. For the 2D dataset distribution, see Figure 5.

4.1 Building the M-Tree

The datasets were indexed in five ways. The single-way insertion method and the bulk loading algorithm (in the graphs denoted as SingleWay and Bulk Loading) represent the original methods of the M-tree construction. In addition to these

Table 1. M-tree statistics.

Metric: L_2 (euclidean)	Node capacity: 20	Dimensionality: 2 – 50
Tuples: 20,000 – 1,000,000	Tree height: 3 – 5	Index size: 1 – 400 MB

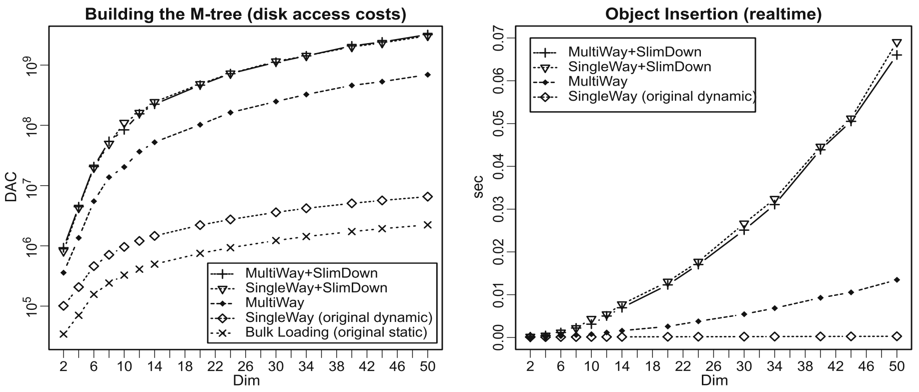


Fig. 6. Building the M-tree: a) Disk access costs. b) Realtime costs per one object.

methods, the multi-way insertion method (denoted as MultiWay) and the generalized slim-down algorithm represent the new building techniques introduced in this article. The slim-down algorithm, as a post-processing technique, was applied on both SingleWay and MultiWay indexes which resulted into indexes denoted as SingleWay+SlimDown and MultiWay+SlimDown. Some general M-tree statistics are presented in Table 1.

The first experiment shows the M-tree building costs. In Figure 6a, the disk access costs are presented. We can see that the SingleWay and Bulk Loading indexes were built much cheaply than the other ones, but the construction costs were not the primary objective of our approach. Figure 6b illustrates the average realtime costs per one inserted object. In Figure 7a, the fat-factor characteristics of the indexes are depicted. The fat-factor of SingleWay+SlimDown and MultiWay+SlimDown indexes is very low, which indicates that these indexes contain relatively few overlapping regions. An interesting fact can be observed from the Figure 7b showing the average node utilization.

The MultiWay index utilization is by more than 10% better than the utilization of the SingleWay index. Studying this value is not relevant for the SingleWay+SlimDown and MultiWay+SlimDown indexes since the “slimming-down” does not change the average node utilization, thus the results are the same as those achieved for SingleWay and MultiWay.

4.2 Range Queries

The objective of our approach was to increase the querying performace of the M-tree. For the query experiments, sets of query objects were randomly selected

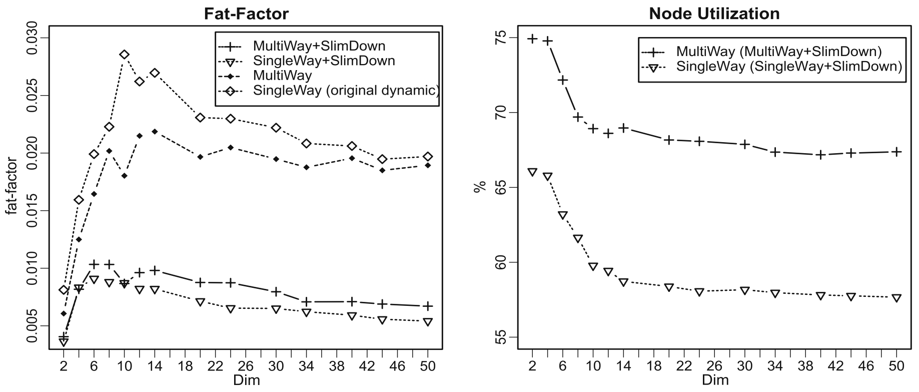


Fig. 7. Building the M-tree: a) Fat-factor. b) Node utilization.

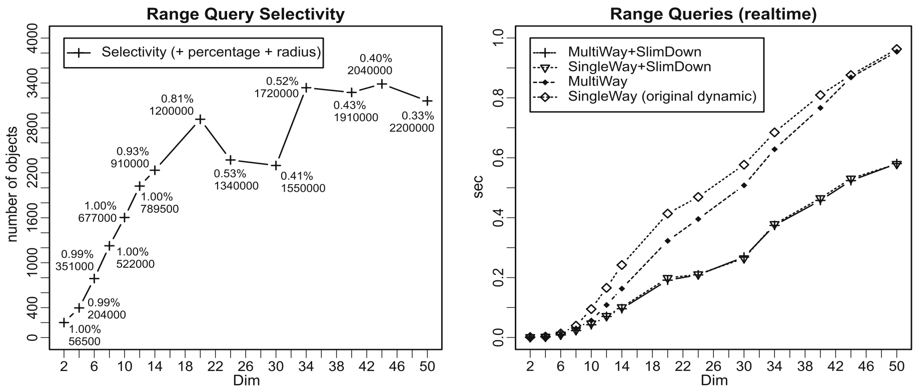


Fig. 8. Range queries: a) Range query selectivity. b) Range query realtimes.

from the datasets. Each query test consisted from 100 to 750 queries (according to the dimensionality and dataset size). The results were averaged.

In Figure 8a, the average range query selectivity is presented for each dataset. The selectivity was kept under 1% of all the objects in the dataset. For an interest, we also present the average query radii. In Figure 8b, the realtime costs are presented for the range queries. We can see that the query processing of the SingleWay+SlimDown and MultiWay+SlimDown indexes is almost twice faster when compared with the SingleWay index.

The disk access costs and the computation costs for the range queries are presented in Figure 9. The computation costs comprise the total number of the d function executions.

4.3 *k*-NN Queries

The performace gain is even more noticeable by the k-NN queries processing. In Figure 10a, the disk access costs are presented for 10-NN queries.

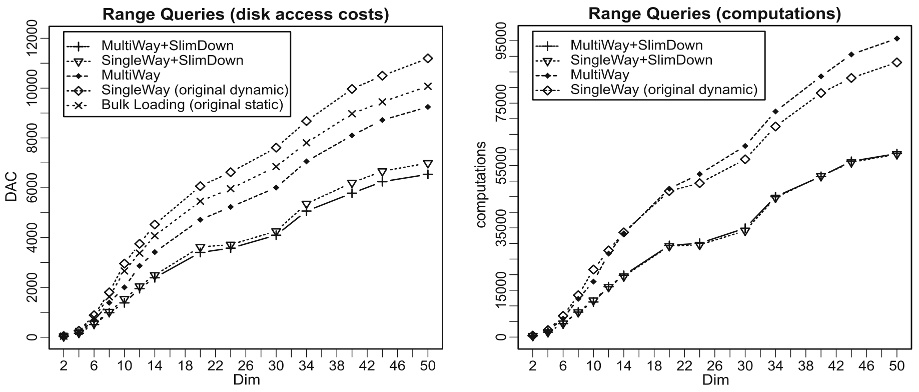


Fig. 9. Range queries: a) Disk access costs. b) Computation costs.

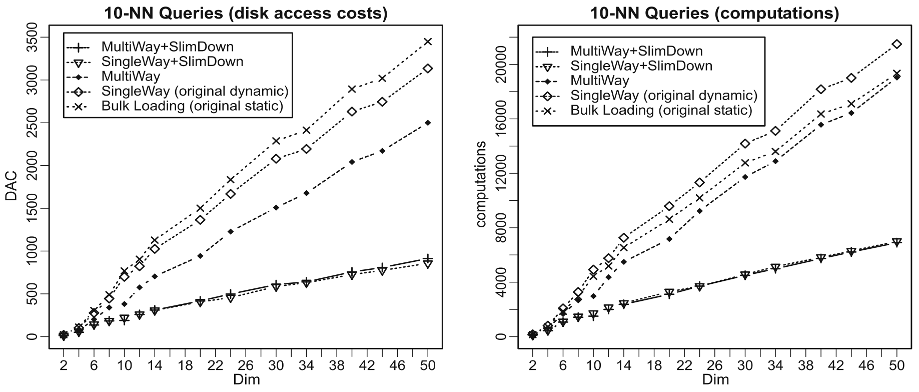


Fig. 10. 10-NN queries: a) Disk access costs. b) Computation costs.

As the results show, querying the SingleWay+SlimDown index consumes 3.5-times less disk accesses than querying the SingleWay index. Similar behaviour can be observed also for the computation costs presented in Figure 10b. The most promising results are presented in Figure 11 where the 100-NN queries were tested. The querying performance of the SingleWay+SlimDown index is here better by more than 300% than the performance of the SingleWay index.

5 Conclusions

In this paper we have introduced two dynamic techniques of building the M-tree. The cheaper multi-way insertion causes superior node utilization and thus smaller indexes, while the querying performance for the k-NN queries is improved by up to 50%. The more expensive generalized slim-down algorithm causes superior querying performance for both the range and the k-NN queries, for the 100-NN queries even by more than 300%.

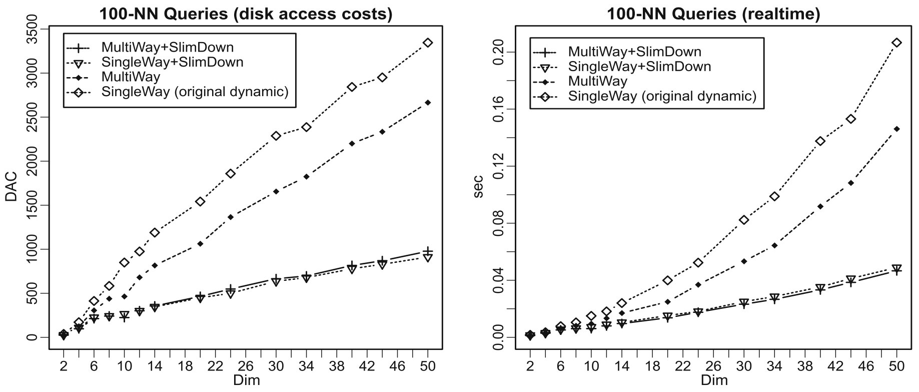


Fig. 11. 100-NN queries: a) Disk access costs. b) Realtime costs.

Since the M-tree construction costs used by the multi-way insertion and mainly by the generalized slim-down algorithm are considerable, the methods proposed in this paper are suited for DBMS scenarios where relatively few insertions to a database are requested and, on the other hand, many similarity queries must be quickly answered at a moment.

From the DBMS point of view, the static bulk loading algorithm can be considered as a transaction, hence the database is not usable during the bulk loading algorithm run. However, the slim-down algorithm, as a dynamic post-processing method, is not a transaction. Moreover, it can operate continuously in a processor idle time and it can be, whenever, interrupted without any problem. Thus the construction costs can be spread over the time.

References

1. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of World-Wide Computing and its Applications'97, WWCA'97, Tsukuba, Japan*, 1997.
2. J. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communication of the ACM*, 18(9):508–517, 1975.
3. S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd Intern. Conf. on VLDB, Mumbai (Bombay), India*, pages 28–39. Morgan Kaufmann, 1996.
4. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
5. T. Bozkaya and Z. M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.
6. P. Ciaccia and M. Patella. Bulk loading the M-tree. In *Proceedings of the 9th Australian Conference (ADC'98)*, pages 15–26, 1998.
7. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.

8. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
9. E. Navarro, R. Baeza-Yates, and J. Marroquin. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
10. M. Patella. *Similarity Search in Multimedia Databases*. Dipartmento di Elettronica Informatica e Sistemistica, Bologna, 1999.
11. H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(3):184–260, 1984.
12. H. Samet. *Spatial data structures in Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 361–385. Addison-Wesley/ACM Press, 1995.
13. C. Traina Jr., A. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. *Lecture Notes in Computer Science*, 1777, 2000.
14. J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
15. P. N. Yanilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms - SODA*, pages 311–321, 1993.
16. C. Yu. *High-Dimensional Indexing*. Springer–Verlag, LNCS 2341, 2002.

Compressing Large Signature Trees

Maria Kontaki, Yannis Manolopoulos, and Alexandros Nanopoulos

Department of Informatics, Aristotle University of Thessaloniki, Greece
{kontaki,manolopo,alex}@delab.csd.auth.gr

Abstract. In this paper we present a new compression scheme for signature tree structures. Beyond the reduction of storage space, compression attains significant savings in terms of query processing. The latter issue is of critical importance when considering large collections of set valued data, e.g., in object-relational databases, where signature tree structures find important applications. The proposed scheme works on a per node basis, by reorganizing node entries according to their similarity, which results to sparse bit vectors that can be drastically compressed. Experimental results illustrate the efficiency gains due to the proposed scheme, especially for interesting real-world cases, like basket-market data or Web-server logs.

1 Introduction

Nowadays, the database sizes continuously increase due to the increase in the size of data. During the previous years, various structures have been proposed for the storage of data in smaller space, but mainly for their more efficient processing [WMB99]. One of these structures is the *signature tree* (S-tree) [Depp86], which is used to index objects with multi-valued attributes. Objects of this type are used in object-oriented databases, in digital library systems, in WWW search engines, or in multimedia databases.

The value of each attribute of an object can be represented by a signature, that is, a bit vector produced by applying a hashing function on the attribute's value. The total number of aces (bits equal to one) is the weight of the signature. An object's signature is produced by superimposing (i.e., OR-ing) each of its attribute signatures. Initially, signatures were used as indices in the signature files that are sequential structures and require the scanning of the entire collection of signatures.

Deppisch [Depp86] proposed the S-tree, which similar to the B+-tree, is a height-balanced tree structure. In the S-tree, a signature of a node at level *i* is produced by superimposing all signatures of its child nodes that at level *i*+1, considering that the root is at level 0. As result, the upper levels have signatures with 'heavy' weight (many aces with respect to the length of signature). Therefore, the selectivity of such nodes reduces and the performance of structure during query processing is affected, since for each query a large number of nodes are retrieved. With primary goal the reduction of signatures weight, improved signature structures were proposed [TNM00].

Due to the way that signature construction is performed (using a hashing function), information loss may be imported: it is possible that two different objects have the same signature and, thus, it is possible to retrieve objects that do not satisfy the query. The latter case corresponds to the *false-drops*. False-drops affect the performance of the structure, and for this reason, several methods of signature construction were developed, which decrease the false-drops probability [Zezu88].

Two factors mainly affect the performance: (a) the node retrieval time (I/O) and (b) the node processing time (CPU). Although progress has been marked in the disk technology and the reduction of access time, it is, however, not as significant as the progress in processor speeds. Thus, factor (a) still has a significant impact. To overcome this problem, the technique of *compression* can be considered as means to decrease the number of disk accesses. Compression has been primarily associated with the reduction of storage space. However, nowadays, this does not anymore comprise a crucial objective, since the problem of storage space is not considered as much intense as the need of performance during query processing. For this reason, the focus in on how compression can reduce the I/O overhead (node retrieval time) by storing the nodes in less disk pages and, thus, reducing their retrieval time. By achieving a good rate of compression, the performance of query processing can be significantly improved, despite the overhead that is added by the additional (CPU) time required by the compression and decompression.

A generalized framework for compressing index structures (e.g. B-tree, R-tree, etc) has been introduced by [Teuh01], which describes two categories of compression: (a) compression of stored information, and (b) compression of pointers in nodes. The effectiveness of the former (a) category depends on the distribution and the number of different values. In this category compression can apply the lossy scheme (which leads to an increase of false drops) or the lossless scheme.

1.1 Contributions and Layout

In this paper, we propose a novel compression scheme for the S-tree, which is based on the aforementioned issues. The proposed scheme is lossless and is applied in the data entries of nodes (stored information), i.e., the (a) category according to the aforementioned framework. In particular, the scheme works on a per node basis by reorganizing node entries according to their similarity (exploiting node clustering), which results to sparse bit vectors that can be drastically compressed. Emphasis is given to the query processing performance. For this reason the scheme contains an efficient decompression method that requires low CPU times, thus it does not compromise the gains due to the reduction of I/O overhead. Moreover, the proposed scheme is based on compression techniques for sparse vectors, which have been studied in depth during the previous years [BK91].

The contributions of this paper are summarized in the following:

- The development of the compression scheme for the S-tree structure, according to the framework of [Teuh01].
- The development of a novel decompression method which, during query processing, avoids the decompression of the entire tree and, moreover, it uses optimisations (e.g., the adjustment of bits in the query according to the bits in the decompressed nodes). As a result, the CPU time required for decompression is kept significantly low.

- Detailed experiment results, with both real and synthetic data, which examine the weight of signatures, the correlation between the signatures (a case that is met often in real-world applications), and the query length.

It must be additionally noticed that although the proposed compression scheme follows the framework described in [Teuh01], the latter describes index structures in general (containing a small discussion on the S-tree, which comprises the motivation in our scheme), whereas its experiments focused on the R-tree. Our scheme proposes a number of significant contributions compared to the aforementioned general framework, namely: (a) the efficient decompression method, that makes feasible the query processing with the compressed tree by drastically reducing the CPU overhead; (b) the detailed examination of specific details with respect to signature data and the S-tree, e.g., the correlation between signatures; and (c) the experimental results that study the effectiveness of the approach on the S-tree structure.

The rest of the paper is organised as follows. Section 2 describes the related work. In Section 3, we present the proposed scheme. Section 4 contains the experimental results, while the conclusions and the future work are given in Section 5.

2 Related Work

The first approach to index signatures was through sequential signature files [CF84]. Although sequential files reduce the cost for searching the data, they have the drawback that all the signatures in the sequential file are probed during the searching.
S-trees have been proposed in [Depp86], to overcome the aforementioned problem. S-trees are height-balanced tree (analogous to B^+-trees) and they organize the signatures according to criteria like the minimization of weight increase (for more details, see Section 2.1). Also, [S-DR87] have proposed a two-level index structure for the efficient organization of signatures.

Several shortcomings of the original S-tree, especially with respect to the node-split policy, where addressed in [TNM00]. Other improvements for the organization of signatures in tree structures can be found in [TBM02,NM02], which also examine different types of queries (e.g., super-set queries, similarity queries, etc). Extensions of the use of signature indexes to several applications are included in [NTVM02, MNT03]

A description of compression schemes that can be applied to tree structures, in general, is given in the [Teuh01]. Experiments in [Teuh01] have focused on the R-tree structure, whereas it also contains a small description regarding the S-tree, which forms the motivation in our approach (for the new contributions of our approach, see the discussion in Section 1.1). A thorough examination of the advantages of compression can be found in the [WMB99]. Finally, [BBJK+00] proposes a compression method for indexes for high dimensional spaces and [GRS98] for relational databases and indexes for relational data. However, these approaches address much different requirements than the ones considered by our approach.

2.1 The S-Tree

S-trees [Depp86], similarly to B^{+}-trees, are height-balanced trees having all leaves at the same level. Each node contains a number of pairs, where each pair consists of a signature and a pointer to the child node. The S-tree is defined by two integer parameters: K and k. The root can accommodate at least two and at most K pairs, whereas all other nodes can accommodate at least k and at most K pairs. Unlike B-trees where $k = K/2$, here it holds that: $1 \le k \le K/2$. The tree height for n signatures is at most: $h = \lceil \log_k n - 1 \rceil$. Signatures in internal nodes are formed by superimposing (OR-ing) the signatures of their children nodes.

Due to the hashing technique used to extract the object signatures, the S-tree may contain duplicate signatures corresponding to different objects. In Figure 1, an example of an S-tree with height h=3 is depicted, where signatures in the leaves represent individual set signatures (i.e. the indexed objects). For simplicity these signatures are assumed to be of equal weight, i.e., $\gamma(s) = 3$, but they vary from 3 to 6 in upper levels due to superimposition.

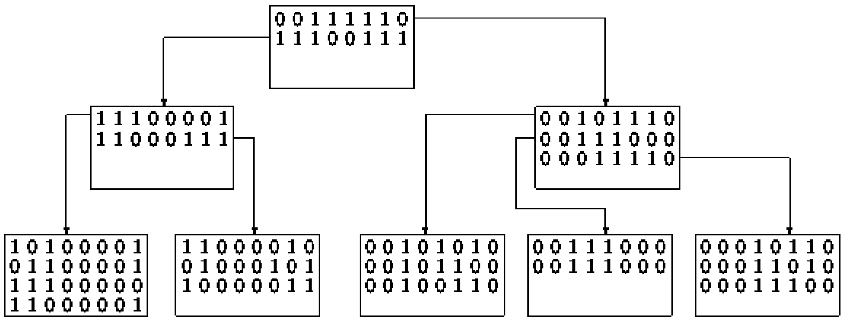


Fig. 1. An example of an S-tree

Successful searches in an S-tree proceed as follows. Given a user query for all sets that contain a specified subset of objects, we compute its signature and compare it to the signatures stored in the root. For all signatures of the root that contain 1s at least at the same positions as the query signature, we follow the pointers to the children of the root. Evidently, more than one signature may satisfy this comparison. The process is repeated recursively for all these children down to the leaf level following multiple paths. Thus, at the leaf level, all signatures satisfying the user query lead to the objects that may be the desired ones (after discarding false drops). In the case of an unsuccessful search, searching may stop early at some level above the leaf level, if the query signature has 1s at positions where the stored signatures have 0s. Due to lack of space, more details (regarding, insertion, deletion, etc.) can be found in [Depp86].

3 Proposed Scheme

Based on the general framework of [Teuh01], we consider two approaches of compressing an entry x of node N: (a) compressing x with respect to the content of the father of node N, and (b) compressing x with respect to the entries that are stored before x in the node N. Both (a) and (b) cases stand exceptions, without of course creating problem in the compression. For instance, in case (a), the root does not have father node, and, in case (b), the first signature of each node does not have any entries that are stored prior to it. Since the use of (a) does not prevent the use of (b), and vice versa, the proposed scheme exploits both of them. The gain from the first approach (a) is that it limits the possible values of x, thus x is stored in a more condensed way. The second approach (b) represents x as $px+\Delta x$, where px it is the value of the previous entry in N and Δx is the difference between x and px. The difference Δx is possible to contain less information than the entry itself, especially when considering the existence of similarity between the entries stored in a node; something that is in general attained by tree index structures, like the S-tree. Therefore, the gain is due to the smaller requirements for the storage space of Δx compared to that of x.

Regarding query execution, we focus on *containment* queries. In terms of signatures, given a query signature q, such queries search for those signatures s in the tree for which it holds that s AND $q = q$ (these queries are also called *subset* queries). Evidently, for the purpose of query processing, a straightforward approach is to first decompress the entire S-tree and then to execute the query. Nevertheless, we develop a different method, which avoids the cost of decompressing the entire tree and concentrates only to the relevant parts of it (i.e., those invoked by the query). In the remainder of this section, we describe in more detail the compression/decompression methods.

3.1 Compression Method

The first step of the method is the compression with respect to the father node. The signature of the father has resulted by applying the OR-ing of the signatures in its children nodes, and thus it is impossible to have any signature in a child node that has ace in a position that the signature of father has zero. Consequently all these zeroes, which "are imposed" by the father, can be omitted.

As described, the second step of the method is the compression with respect to the previous entry in the same node. This step required that consecutive signatures in a node to be as much as possible similar. As a distance measure (i.e., measure of inverse similarity) one can use the broadly used *hamming* distance, that is, the number of bits that the signatures differ. Since the signatures in a node of the S-tree are not ordered, and they are entered in the order they arrive, we can order the signatures within the node so that the total sum of distance between consecutive signatures is minimized.

The aforementioned requirement can be easily transmuted in finding a solution for the problems that belong in the family of travelling salesman problems (TSP). TSP is applied to an underlying graph. In our case, the graph consists of the node's signatures, which correspond to the vertices, whereas the edges are the intelligible lines that imply consecutive signatures. The hamming distance between two signatures (vertices) comprises the weight of the corresponding edge that connects them. Evidently, for large graphs (that correspond to nodes with many entries), one has to resort to one of the several well-known heuristics for the TSP problem. For purposes of simplicity, we used the heuristic that is based on the minimum spanning tree of the graph.

Considering all the aforementioned issues, the compression scheme for a given node, is described as follows:

1. Remove from signatures of the children nodes, the positions that the signature of the father node has zeroes (obvious all the signatures of root are excluded from this step).
2. Apply a heuristic for the travelling salesman problem. The result of the heuristic will be an ordering of the node's signatures, which results to a small sum of Hamming distances between the consecutive signatures within this ordering.
3. Store the first signature of the resulting ordering, as it is. For each of the following signatures, calculate the XOR result with its previous signature (by applying the XOR logical operator between the bits in corresponding positions).
4. The signatures that result from step 3 are sparse vectors (due to the minimization of hamming distance). Apply a compression of the resulting sparse vectors (to be explained in the following).

To perform the compression in the entire S-tree, we have to apply the previously described algorithm (that works on a per node basis) to each of its node. The corresponding algorithm must be applied in a post-order tree traversal. This way of traversal is required due to the need of knowing the bits of the father's signature that are equal to 0, when compressing the signatures in its children. Initially, the root node is excluded from the compression. Then, for each node, we apply the node compression to its children nodes, store them, and then apply it also to the node itself. For a more clear description, the 3rd step of the node compression method is exemplified in Fig. 2.

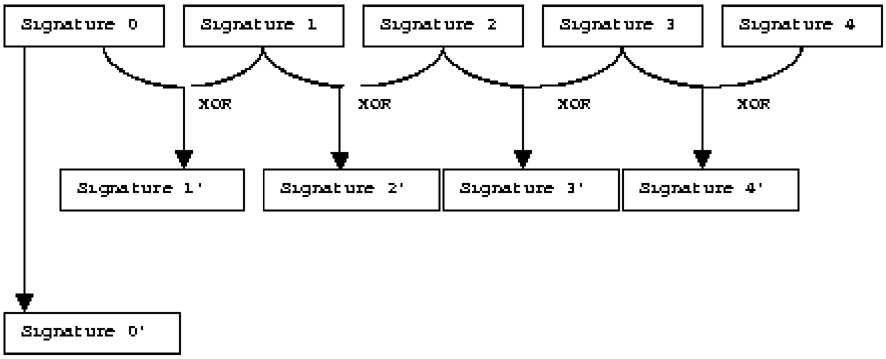


Fig. 2. An example of step 3 in the S-tree node compression scheme

For the 4th step of the node compression method, in our implementation we used a simple way of compressing sparse vectors. Consider that l is the length of signature and s is the number of aces in the signature (i.e., the weight of signature). For sparse vectors it holds that $s << l$. We mark the bit positions that are equal to 1. For the storage of a position, $d=|\log_2 l|$ bits are required. Therefore, for each signature, $s \times d$ bits are needed. Evidently, the use of more sophisticated methods for compressing sparse vectors will lead to improved compression of the entire structure. This is left as a topic of future work, since herein we are interested in testing the effectiveness of the proposed scheme regardless of the specific method used for sparse vectors (i.e., we would like test its viability even for simple such methods). Important is the observation that, with respect to the proposed compression scheme, the resulting signatures do not have the same length (even in the same node), due to the compression of sparse vectors. Therefore, along with each sparse vector, we also store the length of the resulting signature (in Step 4 of the node compression method).

Finally, we have to notice that, in the uncompressed S-tree, each node is stored in an entire disk page, something that will lead to space wastefulness in the case of the compressed S-tree. The reason is that a compressed node occupies smaller space compared to an uncompressed one. Therefore, at each disk page, we store as many (compressed) nodes as possible. In our implementation, for reasons of simplicity, we considered that a node does not span different pages, thus a small fragmentation may incur. Nevertheless, if we do not use this simplification, the performance gains are expected to increase further.

3.2 Basic Node Decompression Method

As described, a straightforward method for processing queries would be to apply the reverse process of compression for the entire tree and, then, to execute the query over the decompressed tree. However, this would not be effective for two reasons: (a) no gain can be reaped, because the latter part in this procedure is equivalent to the querying of the tree when compression is not applied at all; (b) the total cost is burdened by the additional cost to uncompress the tree. Obviously, due to (a) and (b), the total performance would be worse than in the case where compression is not used.

Thus, in the proposed scheme, only the required part of the tree (the nodes invoked by the query) is involved during decompression, which is performed simultaneously with the processing of the query. Therefore, the overhead of decompression is decreased drastically.

As in the compression procedure, herein we will first describe a node decompression method and then the complete decompression algorithm. The first step in the node decompression method is the decompression of sparse vectors: we compute the number d (number of the bits that is required for the storage of a bit with value equal to one in an uncompressed signature) and, moreover, we initialize the new signature (all bits are initially set to 0). We divide the signature into groups of d bits. We find the positions of aces and set them to 1 (putting 1 to the corresponding bit of the new signature), turning each such group into a decimal number. It has been observed (by our experiments) that this is the most time-consuming step in the whole decompression method.

The next step of the node decompression method is to "inverse" the XOR operation between the consecutive signatures. We begin from most leftmost (first) signature of the node, in which the XOR operation was not applied so as to constitute our base for the inversion. For each pair of consecutive signatures, we apply the XOR operation (notice that by XOR-ing twice, we get the inverse) and, thus, we take the initial signatures. In Figure 3 is given an example that clarifies the previous procedure.

Finally, the node decompression method is completed by adding to the node's signatures the bits of the father's signature, which are equal to 0.

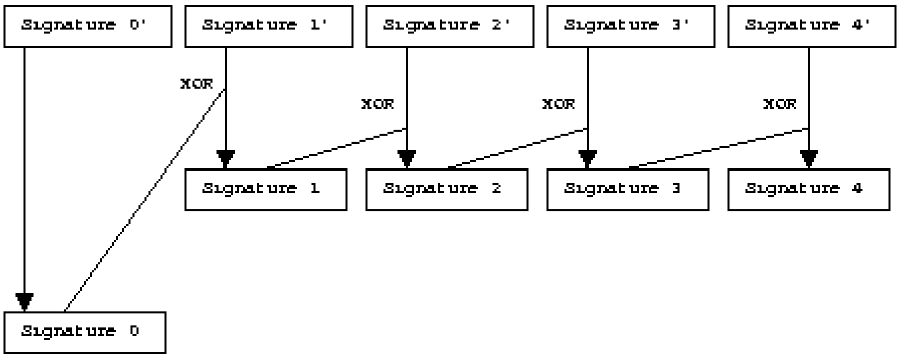


Fig. 3. An example of node decompression

3.3 Improved Decompression Method

In the basic node decompression method, the last step (i.e., the addition of bits according to the ones in the father's node signature, which have value equal to 0), overloads by far the decompression time, since it has to be applied in each signature of the node. To overcome this problem, we can omit the corresponding bits of the father's signature (i.e., those equal to 0) from the query signature. Please recall that the decompression is executed at the same time with the query. For example, let a node containing 90 signatures and its father's signature has 15 bits equal to 0. Therefore with the basic process, we should add 1350 bits, in total, to the signatures of the node. In contrast, with the method that was previously described, we entirely avoid this cost. It has to be noticed that the omission of the bits from the query signature is performed only for the signatures in the subtree of the father's signature (i.e., at each node, a local copy of the query's signature is used). From this, it is easy to see that the correctness of the query result is not affected.

Regarding the complete decompression method, different from the case of compression, the node decompression algorithm is applied through a *preorder* traversal of the S-tree. This is because it is necessary to first decompress the father-node before we continue with his children, so as to know the bit positions that are equal to 0. The decompression method considers as basic unit the node (i.e. when one node is selected, all his signatures decompress). This is contrast to the approach of [GRS98], which considers as basic unit each individual entry in an index structure.

4 Performance Study

In this section, we present the experiment results that measure the performance gains due to the proposed scheme. We have conducted experiments to measure the compression rate (i.e., ratio of size between the compressed and uncompressed cases). However, due to space constraints, we herein focus on the measurement of performance during query execution, since, as described, this measure corresponds to the case of interest. The compressed S-tree is denoted as COMP and the uncompressed as ORIG. Next, we first describe the experimental setup and then we present the results.

4.1 Experimental Setup

For both COMP and ORIG we used the improved construction methods that were proposed in [TNM00]. For each experiment we present two diagrams: (a) one for the comparison between COMP and ORIG in terms of the disk accesses required by the query, and (b) one for the CPU time (measured in seconds) required by the decompression during the query execution (this time is presented only for COMP, since ORIG does not require it). The number of the disk accesses does not result from the count of the actual disk accesses but from the count of disk pages accesses and this holds for all experiments. As mentioned, we focus on the containment (subset) queries, which are measured with respect to the items involved in the query signature. The default page size is 8 K. The experiments were conducted on a PC with processor AMD Athlon at 1.6 GHz.

We used both real and synthetic data sets, with various values of weight/length of signatures. Hereafter, s denotes a signature, F its length, and with $w(s)$ its weight (the number of aces that contains). In addition, we use the notion of *fraction weight*, denoted as $wF(s)$, in order to express the number of bits in s that are equal to 1 with respect to its length (e.g., if F is equal to 512 and $w(s)$ is 256, then $wF(s)$ is equal to 0.5). For synthetic data, a fraction of the number of bits that were equal to 1 in a signature were also set to 1 in the immediately next generated signature. This yields to a correlation between the generated signatures. We have observed this in several real-world cases (e.g., basket market data), which are in contrast to the non-realistic assumption of independent bit values within signatures. Correlation is characterized by the aforementioned fraction, which is denoted as correlation factor (*corF*).

The real data that we used are two WWW traces, namely: (a) the ClarkNet data set, that contains two week's worth of all HTTP requests to the ClarkNet WWW server, and (b) the NASA data set, that contains two month's worth of all HTTP requests to the NASA Kennedy Space Center.[1] From these traces, we extracted user sessions, where each one was represented by a signature. The number of signatures for the ClarkNet data set is about 75,000 and the distinct items are 7200. For the NASA data

[1] The ClarkNet and NASA data sets can be obtained from http://ita.ee.lbl.gov/html/traces.html.

set, the number of signatures is equal to 100,000 and the distinct items are 1800. Both data sets present the aforementioned characteristic of correlation.

The performance metrics we used were: i) the accessed tree nodes, and ii) the compression ratio. Regarding the former (i) we did not employ a buffering scheme, since we wanted to focus on the complexity of query execution in terms of the number of nodes invoked during a query. Nevertheless, it is expected that the use of buffering will have the equivalent impact on all methods (thus, the relative performance difference will be preserved).

4.2 Experimental Results

In the first experiment, we examine performance for synthetic data with respect to the size of query, which is given in terms of its $wF(q)$. For each data signature s, its length F is equal to 512 and its $wF(s)$ is 0.05. The correlation factor was set up equal to 0.5. For the experiment we used an S-tree with a maximum of 120 signatures per node and with a minimum of 40 (denoted as 120/40), whereas the number of signatures were 50,000. The results are depicted in Figure 4. More specifically, the left part of Figure 4 depicts the number of disk accesses (denoted as DA), whereas the right part of Figure 4 depicts the CPU overhead due to decompression (for COMP only). As shown, a clear reduction in the number of disk accesses is attained by COMP. Moreover, the cost of decompression is low enough to guarantee that the total cost of query execution for COMP is much smaller than that for ORIG. As $wF(q)$ increases, the number of disk accesses is reduced for both COMP and ORIG. This is as expected, because with increased $wF(q)$ less data signatures satisfy the query and, thus, less nodes are examined (analogous reasoning can be followed for the reduction of the decompression time for COMP).

The second experiment is similar to first, however using different values for certain parameters. The factor weight wF(s) is equal to 0.9 and the correlation factor is equal to 0.3. The values of other parameters remain the same as in the previous experiment. The left part of Figure 5 shows the number of disk accesses for the COMP and the ORIG and right part of Figure 5 shows the time of decompression for the COMP. As shown, in this case, again, the number of disk accesses of COMP is less than that of ORIG (the decompression time is also quite low). Comparing the two experiments, however, we observe that at the second experiment the number of disk accesses for both methods is significantly increased. Moreover, the number of disk accesses is not decreased with increasing query factor weight. This is as expected, due to the high 'heavy' data signatures that were used, which lead to the creation of data signatures with low selectivity. Only in the case where the query factor weight is very high, number of signatures that satisfies the query (and the number of disk accesses) is relatively decreased.

In the following experiment we examined a different type of synthetic data sets to model basket market data, i.e., customer transactions, which were produced using the generator described in [AS94]. The characteristics of these data are that their signatures are sparse (they have few aces with respect to the length of the signature) and that the correlation between them is high (i.e., a case analogous to the one examined in the first experiment). Because the queries were generated from the data (to follow their distribution), the performance is examined with respect to the absolute weight of a query signature $w(q)$ instead of the query weight factor $wF(q)$. Following the notation of [AS94], the used dataset was the T10.I6.D100K, thus the number of data signatures was 100,000, the average number of items per customer transaction was equal to 10, and the other parameters were the default ones used by the generator. The results are illustrated in Figure 6. The left part of Figure 6 illustrates that COMP significantly outperforms ORIG in terms of required number of disk accesses, whereas the decompression time (right part of Figure 6) is low enough. This result indicates that for such interesting real-world cases, the proposed scheme can attains significant performance improvements, due to the characteristics of the data.

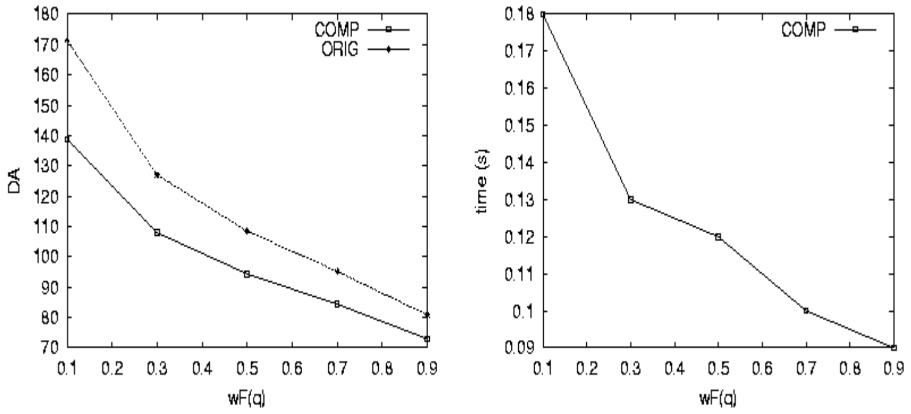


Fig. 4. Query performance with respect to query size for synthetic data. *Left*: Disk accesses w.r.t. *wF*(*q*). *Right*: Decompression time

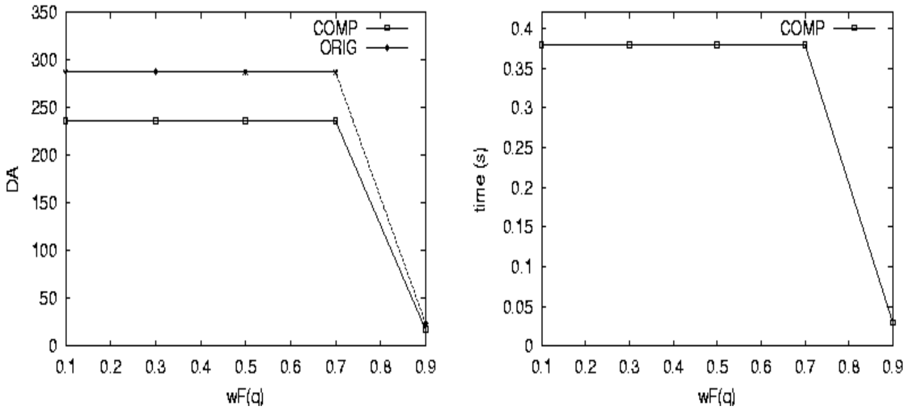


Fig. 5. Query performance with respect to query size for synthetic data. *Left*: Disk accesses w.r.t. *wF*(*q*). *Right*: Decompression time

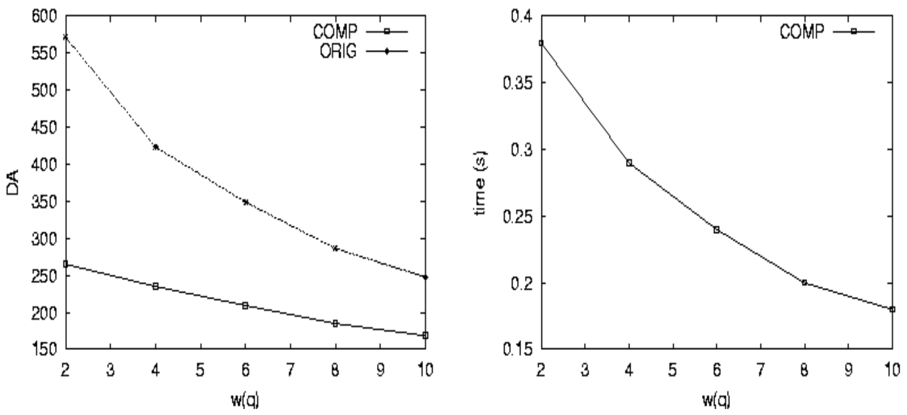


Fig. 6. Query performance with respect to query size for market basket data. *Left*: Disk accesses w.r.t. *w*(*q*). *Right*: Decompression time

Also, we conducted two more experiments with real data sets, in order to verify the measurements obtained with synthetic data. Similar to the basket-market case, the queries were generated from the data signatures and their length is given with the *w*(*q*) measure.

The results for the ClarkNet data set are illustrated in Figure 7. We observe that the disk accesses (left part of Figure 7) and the decompression time (right part of Figure 7) are similar with those in previous experiments. COMP clearly outperforms ORIG for all query sizes. The results for the NASA data set are depicted in Figure 8. As shown, for small queries (*w*(*q*) less than 4) COMP is significantly better than ORIG. For higher weights, however, both methods require similar disk accesses, since the data signatures in this case are very sparse and the number of signatures that satisfy the query is very small. Therefore, compression may not pay-off for this cases, considering that, although small, the additional cost of decompression has to be paid.

Table 1. Compression ratios

Data set	**S-tree**	**Compressed S-tree**	**Compression Ratio**
Synth: wF(s)=0.05, corF=0.5	3155 KB	1798 KB	43%
Synth: wF(s)=0.9, corF=0.3	3162 KB	2964 KB	6.3%
Market basket data	5894 KB	1711 KB	71%
ClarkNet	6333 KB	794 KB	87.5%
NASA	4762 KB	911 KB	80.9%

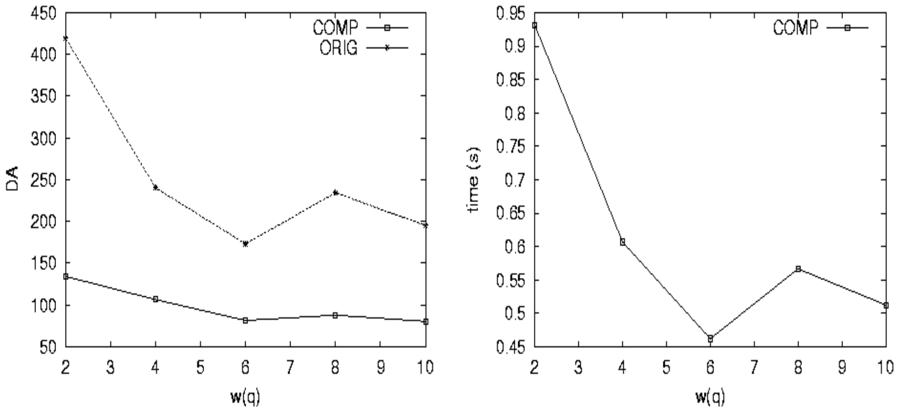


Fig. 7. Query performance with respect to query size for the ClarkNet data set. *Left*: Disk accesses w.r.t. *w*(*q*). *Right*: Decompression time

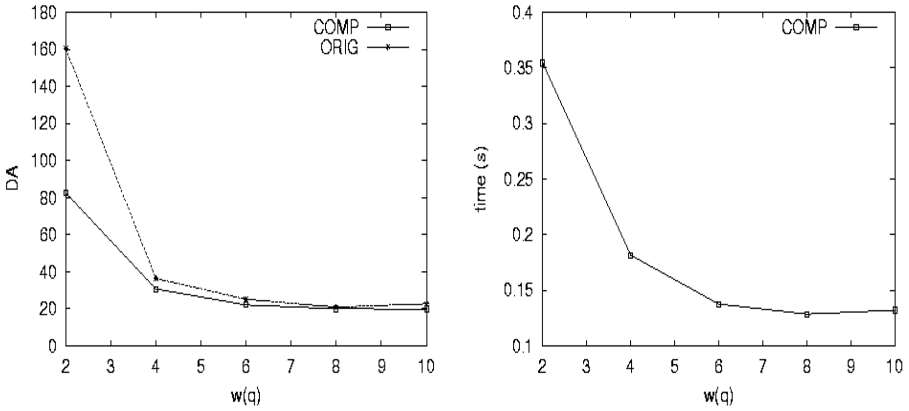


Fig. 8. Query performance with respect to query size for the NASA data set. *Left*: Disk accesses w.r.t. *w*(*q*). *Right*: Decompression time

5 Conclusions and Future Work

We have examined the problem of compressing large signature tree structures aiming to significantly improvement in performance during query processing. We proposed ascheme that consists of the corresponding compression and decompression method. Both methods work on a per node basis and are based on the grouping of node entries according to their similarity, the production of sparse vectors, and the compression of the latter. The decompression method overcomes the problems of the straightforward approach (i.e., the decompression of the entire tree) and attains low overhead for this operation.

Detailed experimental results with real and synthetic data illustrated the gains reaped due to the proposed scheme. Especially for interesting real-world cases (basket market data, real traces), the proposed scheme presents significant improvements in terms of I/O cost, and very low CPU time overhead for decompression.

Future work includes the examination of more complicated techniques for the compression of sparse vectors [BK91]. Also, we will focus on the development of compression schemes that will support the construction of dynamic S-trees. The naive technique is the decompression of entire tree, the insertion of a signature and the compression again of entire tree. This technique is not effective. An efficient algorithm, similar with the one that we follow in the stage of decompression, is: decompress only the nodes that are invoked in the insertion and are determined from the insertion algorithm, which is executed simultaneously. More gain can be achieved by the observation that if the father of a node is the same before and after the insertion then all the nodes above this node (i.e., the nodes more close to the root) will be the same and therefore it is not need to compress them again. Moreover a great overhead is introduced in the stage of node split and therefore the criteria of signature insertion should be selected in order to minimize the number of splits. Further issues should be researched.

References

[AS94] R. Agrawal, R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". *Proc. Conf. On Very Large Databases (VLDB'94)*, pp. 3–14, 1995.

[BBJK+00] S. Berchtold, C. Bohm, H. Jagadish, H.-P. Kriegel, J. Sander. "Independent Quantization: an Index Compression Technique for High Dimensional Data Spaces". *Proc. Conf. on Data Engineering (ICDE'2000)*, pp. 577–588, 2000.

[BK91] A. Bookstein, S. Klein. "Compression of Correlated Bit-Vectors". *Information Systems*, Vol.16, No.4, pp.387–400, 1991.

[CF84] S. Christodoulakis, C. Faloutsos. "Signature Files: An Access Method for Documents and its Analytical Performance Evaluation". *ACM Transactions on Office Information Systems*, Vol. 2, pp. 267–288, 1984.

[Depp86] U. Deppisch. "S-tree: A Dynamic Balanced Signature Index for Office Retrieval". *Proc. ACM SIGIR Conf.*, pp. 77–87, 1986.

[GRS98] J. Goldstein, R. Ramakrishnan, U. Shaft. "Compressing Relations and Indexes". *Proc. Conf. on Data Engineering*, pp. 370–379, 1998.

[MNT03] Y. Manolopoulos Y., A. Nanopoulos, E. Tousidou. "*Advanced Signature Indexing for Multimedia and Web Applications*", The Kluwer International Series on Advances in Databases Systems, Kluwer Academic Publishers, 2003, in print.

[NM02] A. Nanopoulos, Y. Manolopoulos. "Efficient Similarity Search for Market Basket Data". *The VLDB Journal*, Vol. 11, No. 2, pp. 138–152, 2002.

[NTVM02] M. Nascimento, E. Tousidou, C. Vishal, Y. Manolopoulos. "Image Indexing and Retrieval Using Signature Trees". *Data and Knowledge Engineering*, Vol. 43, No. 1, pp. 57–77, 2002.

[S-DR87] R. Sacks-Davis, K. Ramamohanarao. "Multikey Access Methods Based On Superimposed Coding Techniques". *ACM Transactions on Database Systems*, Vol. 12, No. 4, pp. 655–696, 1987.

[Teuh01] J. Teuhola. "A General Approach to Compression of Hierarchical Indexes". *Proc. Database and Expert Systems Applications (DEXA'2001)*, pp. 775–784, 2001.

[TBM02] E. Tousidou, P. Bozanis, Y. Manolopoulos. “Signature-based Structures for Objects with Set-valued Attributes”. *Information Systems*, Vol. 27, No. 2, pp. 93–121, 2002.

[TNM00] E. Tousidou, A. Nanopoulos, Y. Manolopoulos. “Improved Methods for Signature-Tree Construction”. *The Computer Journal*, Vol. 43, No. 4, pp. 301–314, 2000.

[WMB99] I. Witten, A. Moffat, T. Bell. “*Managing Gigabytes – Compressing and Indexing Documents and Images*”. Morgan Kaufmann, 1999.

[Zezu88] P. Zezula. “Linear Hashing For Signatures Files”. *Proc. IFIP TC6 and TC8 Symp. on Network Information Processing Systems*, pp. 192–196, 1988.

A Conceptual Graphs Approach for Business Rules Modeling

Irma Valatkaite and Olegas Vasilecas

Information Systems Scientific Laboratory
Vilnius Gediminas Technical University
Sauletekio al. 11, Vilnius, Lithuania
{Irma,Olegas}@isl.vtu.lt

Abstract. Business knowledge is the major asset of business organizations, which empowers them not only to survive, but also make proactive decisions in rapidly changing business situation. Therefore mapping business knowledge to the computable form is the primary task for information systems research. Structuring business knowledge using business rules approach is convenient for both business representatives and system analysts. However, the question of modeling language is still open and different approaches are discussed in the literature. In this paper, conceptual graphs are considered as one of the suitable modeling languages. It is proposed to add a new element *rule base* to conceptual graphs element *knowledge base* for explicit business rules modeling in order to satisfy the important requirement of business rules systems – business rules must be addressed explicitly. The different business rules classification schemas are discussed with regard to the proposed business rule metamodel and it is shown how to map business rules regardless of their type to the uniform format.

1 Introduction

Business rules are a modern approach to information systems (IS) design, which addresses business rules explicitly in design, implementation, and maintenance phases. Traditional system development methodologies generally address rules only in the implementation phase and business rules get scattered throughout the system without any tracking possibilities. However, today business organizations must take fast proactive decisions towards rapidly changing business environment and it is vitally important to have a valid business rules set every day. Since business rules change together with business environment, it is unacceptable to wait until implemented business rules set will be adapted if IS was developed using some traditional methodology. The remedy is the rule-based IS development methodology [10].

IS development process starts from conceptual domain models – data model, process model, object model, etc. The type and number of models depend on the methodology used. However, each model must be developed using some modeling language. For the rule-based IS development process the choice of a modeling language for the business rules model is still an open question. A number of languages were proposed, but there is no consensus yet.

In this paper conceptual graphs are considered as one of the suitable modeling languages which offers a number of advantages. Section 2 outlines the requirements for a business rules modeling language and gives a brief overview of proposed modeling techniques. Conceptual graphs and business rules modeling using CG are presented in Section 3 including the proposed business rules metamodel and the rationale behind it. Finally, the discussion, conclusions, and future work directions are outlined in Section 4.

2 Modeling Languages for Business Rules

What are the requirements for a business rules modeling language? This is the question which must be answered before attempting to motivate any choice. We argue that the requirements stem from three basic rules model use cases:

1. *As means to capture and represent business knowledge.* A business rules model as a business knowledge representation must be precise and unambiguous in order to utilize the model for the business rules implementation or as a business knowledge repository. The characteristics of unambiguity and preciseness inherently belong to formal knowledge representation languages based on logic.
2. *As an input for automatic rules processing engines.* Having business rules model in a formal logic-based language makes it possible to use some automatic implementation mechanisms or inference engines. For this purpose the modeling language should have some standard interchange format in order to exploit the model using automatic processing mechanisms. This possibility would greatly facilitate business rules implementation in business information systems.
3. *As a communication basis for business people and system analysts.* Business rules are owned by business people. This fact implies that business people must make the active contribution to the modeling process. Therefore the modeling language used must be easily comprehensible and intuitive. Since pictures in general are more comprehensible than text or formulas, graphical notation is preferable.

A number of languages and approaches were considered for the business rules modeling.

Nowadays the most popular modeling language is UML (Unified Modeling Language) which was created by joint efforts of researchers and commercial organizations [3]. It was proposed to use UML for explicit business rules modeling as well – mostly it is advised to accompany UML with expressive power of OCL (Object Constraint Language) [14] because UML itself does not have any special means to express business rules explicitely. The main advantage of this approach is that UML is a *de facto* standard modeling language with a number of supporting tools. In [2], [6] business rules are expressed in OCL statements and automatic implementation techniques are presented. In [6] generation of SQL views and trigger templates for each rule is presented, but the trigger action part is not automatically generated. A method to generate triggers for consistency rules modeled using OCL is presented in [2], but the authors limit the usage of the method to consistency rules only. While UML with OCL satisfy the requirements of expressiveness, preciseness and unambiguity, they have one significant drawback – although OCL is an expressive formal language, it does not have any graphical notation and thus does not account for the easily comprehensible language.

The Ross method presented in [10] is both a language and a method. Ross has created the original graphical notation to represent business rules in a data model. Specific constructs are defined for each of the rules families together with a big number of accompanying constructs, such as special symbols, invocation values, special interpreters, and special qualifiers. The rules classification schema and constructs present the truly exhaustive method to model business rules. However, a big number of modeling constructs makes the language quite complicated, at least for inexperienced users, and Ross does not define any interchange format for the rules model representation and interchange.

Commercial organizations, such as Oracle, also present their own methods and languages. In [4] CDM RuleFrame environment is presented where special OCL subset called RuleSLang is developed to represent business rules. Later this representation is used for automatic rules enforcement using Oracle technologies. The main drawback of this approach is the lack of the graphical notation (the same as with pure OCL) and the tight coupling with commercial products of one vendor.

Of course, systems of logic (e.g., predicate calculus) can also be considered as possible candidates. While the expressive power is not questionable, they do not account for easily comprehensible and intuitively used languages as well.

Two conclusions may be drawn out this short survey. First, the requirements for a modeling language are high and contradictory therefore it is difficult to find a single modeling language to satisfy them all. Second, there is still a gap between requirements and proposed solutions and the new candidates are welcome.

We propose to use conceptual graphs as a business rules modeling language. The short description of conceptual graphs language, the modeling method and the motivation behind it follows below.

3 Business Rules Modeling Using Conceptual Graphs

3.1 Conceptual Graphs

Conceptual graphs (CG) created by Sowa [11] are a knowledge representation language defined as an extension of C. S. Pierce existencial graphs with features adopted from linguistics and artificial intelligence. Due to the rich set of available formats CG can be used at different levels. At a conceptual level it can be a basis for a specialized communication language between systems analysts and business representatives involved in a common cognitive work because of simple graphical and linear notations. At an implementation level it can be the basis for a common representation tool used by several modules (e.g. knowledge and databases, inference engines) [5] because of direct mapping to first order predicate logic and standard interchange formats (KIF – Knowledge Interchange Format and CGIF – Conceptual Graphs Interchange Format). Since CG are considered as a graphical interface to a system of logic, CG can be used as the basis for connectivity among natural language, formal language, and visual language.

CG were successfully applied to knowledge modeling. For example, in [9] CG were chosen as a modeling language to represent organizational knowledge in corporate memory called Method Repository. When compared to UML, PIF (Process Interchange Format), and Workflow Management Coalition Workflow Reference Model for the task of representing methods, know-how and expertise of the consultants, CG appeared to be the best response to the requirements of corporate memory development.

Thus CG satisfies the requirements for business rules modeling language as they were stated in Section 2: CG are formal logic-based language with several interchange formats: simple graphical notation for human communications, KIF or CGIF for process or engines communications.

The formal structure of CG is defined by the following ten definitions (refer to [12] for full definitions):

1. A *conceptual graph* is a bipartite graph that has two kinds of nodes called concepts and conceptual relations which are linked by arcs. Every arc must link a concept and a conceptual relation in a conceptual graph. Nor two concepts neither two conceptual relations can be linked directly. There can be concepts which are not linked to any conceptual relation, but every arc that belongs to any conceptual relation must be attached to exactly one concept in a conceptual graph. The simple example of a conceptual graph representing the sentence *A cat is on a mat* (a famous example sentence in the CG literature) is shown in Figure 1 using graphical notation.
   The conceptual graph from Figure 1 in linear form and predicate calculus is presented below:
   $[Cat]- > (On)- > [Mat]$
   $(\exists x : Cat)(\exists y : Mat)On(x, y)$
2. A *concept* has a type and a referent. In Figure 1 two concepts are present: *Cat* and *Mat*, where the words "Cat" and "Mat" are types and referent fields are blank. Blank referent fields indicate the existential quantifier.
3. A *conceptual relation* has a relation type and a nonnegative integer called its valence. In Figure 1 *On* is a conceptual relation with valence 2, its signature is $< Cat, Mat >$.

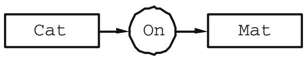


Fig. 1. Conceptual graph for the sentence *The cat is on a mat*

4. A n-adic *lambda expression* is a conceptual graph, called the body of lambda expression, in which n concepts have been designated as formal parameters. E.g. dyadic lambda expression for a sentence *John is going to Boston*:
   $$[Person : \lambda 1] < -(Agent) < -[Go]- > (Dest)- > [City : \lambda 2].$$
5. *Concept Types* are organised in type hierarchy which is a partially ordered set T whose elements are called type labels. Every type label is specified as primitive or defined. For every defined type label there exists a monadic lambda expression called its definition.
6. *Relation Types* are organised in relation hierarchy which is a partially organised set R whose elements are called relation labels. Every relation label is specified as primitive or defined. For every defined relation label there exists n-adic lambda expression called its definition.
7. The *referent* of a concept is specified by a quantifier and designator. The referent of a concept determines the entity or a set of entities the concept refers to. The designator determines the referent by showing its form, pointing to it or describing it. Quantifier determines the quantities or amounts.
8. A *context* is a concept whose designator is a nonblank conceptual graph.
9. A *coreference set* in a conceptual graph g is a set of concepts selected from graph g or from graphs, nested in contexts of g.
10. A *knowledge base* is a context of type KnowledgeBase whose designator is a conceptual graph consisting of four concepts:
    - *Type hierarchy.* A context of type TypeHierarchy whose designator is a conceptual graph T that specifies a partial ordering of type labels and the monadic lambda expressions for each defined type label.
    - *Relation hierarchy.* A context of type RelationHierarchy whose designator is a conceptual graph R that specifies a partial ordering of relation labels, the valences of the relation labels, and the lambda expressions for each defined relation label.
    - *Catalog of individuals.* A context of type CatalogOfIndividuals whose designator is a conceptual graph C that contains exactly one concept for each individual marker that appears in any concept in the knowledge base. The designator may contain other concepts and relations that describe the individuals.
    - *Outermost context.* A context of type Assertion whose designator is a conceptual graph O.

3.2 Business Rules Modeling Technique

In order to model business rules using the knowledge base framework as defined by Sowa, additional constructs are needed in order to model business rules explicitly. Defined knowledge base elements are enough to represent domain structure using concept types hierarchy, relation types hierarchy, concept types and relation types definitions. We propose to add a novel element for the explicit business rules modeling called *rule base*.

Rule base is defined by the following two definitions:

Definition 1. *Type Rule is a conceptual graph g which has a concept of type Event as initiator and a concept of type Conditional Action in the form of if–then rule as a result:*

$$\begin{aligned}&Type\ Rule(*x) \text{ is } T(?x)-\\&\quad < -(INIT) < -[Event : *y]\\&\quad - > (RESULT)- > [If : *z[Then : *w]]\end{aligned}$$

Definition 2. *Rule base is a context of type RuleBase whose designator is a set B of conceptual graphs of type Rule. B specifies rules to be enforced on concept types in T and relation types in R.*

Type *Rule* is a metamodel of a business rule in the knowledge base. It specifies that each rule must have the initiator of type *Event* and the result of type $[If : *z[Then : *w]]$. The latter is the special construct in CG defined for *if–then* rules.

For example, the business rule *Customers which buy for more than $1000 at once are considered as valuable* would be represented the following way:

$$\begin{aligned}&Type\ ValuableCustomer(*x) \text{ is } Rule(?x)-\\&\quad < -(INIT) < -[Event : [Customer : *y]- > (buy)- > [TotalAmt : *z]]\\&\quad - > (RESULT)- > [If : [TotalAmt :?z]- > (>=)- > [Number : 1000]\\&\qquad [Then : [Customer :?y]- > (has)- > [Characteristic : Valuable]]]\end{aligned}$$

The same rule in the graphical notation is shown in Figure 2.

The business rules metamodel closely resembles the ECA rules paradigm from active databases [1]. The key characteristic of active databases is the reactive behaviour which is based on the concept of ECA rule. ECA rule consists of event, condition, and action (hence the name ECA). The meaning of such a rule is: when *event* occurs, if *condition*, then *action*.

These three parts are present in the metamodel as well: the rule initiator is an event (concept of type Event) and the rule result is a conditional action which consists of two parts – *if-part* which represents condition and *then-part* which represents action.

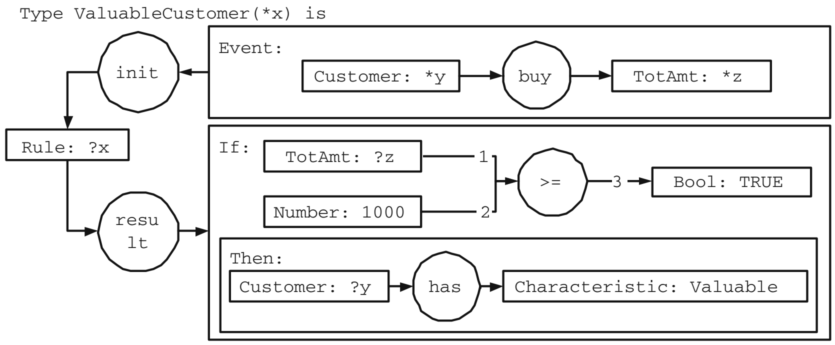


Fig. 2. Business rule *Customers which buy for more than $1000 at once are considered as valuable* in CG graphical notation

This close resemblance is by no means accidental. The usage of active databases and ECA rules (implemented as triggers in active databases) is not a new idea, but nevertheless very much suitable because of two reasons. First, the main characteristic of active databases is the reactive behaviour to various events. According to the definition of active database, it is equipped with means to specify rules, then trigger their execution upon the occurrence of relevant events. Namely these two mechanisms are the key characteristics that provide the functionality required to define, create, manage and use business rules. Second, business rules semantics are very much similar to the semantics of ECA rule: business rule, just like ECA rule, defines under what circumstances certain actions must be carried out, under what circumstances certain actions are forbidden, and what events trigger the rules execution.

3.3 Business Rules Metamodel and Different Rules Classification Schemas

Using the proposed simple yet powerful business rule metamodel it is possible to represent business rules despite of their type. Up to now several business rules classification schemas were proposed which divide business rules to families, types or categories. Ross method [10] and the GUIDE project [7] propose probably the most extensive classification schemas, while Herbst et al. in [8] divide business rules merely to three categories (Herbst schema and Ross business rules families are shown in Figure 3).

Since business rules metamodel uses the ECA paradigm, we will show how all business rules despite of their diversity can be mapped to the ECA paradigm.

All business rule possess three key characteristics: they are data-driven, event-driven and provide a result. Therefore ECA paradigm can be considered as the most general approach for business rules regardless of the classification schema:

– Each rule has an event which cause the rule execution. E.g., in Ross method the event is concealed under the concept of *anchor* which is defined as the

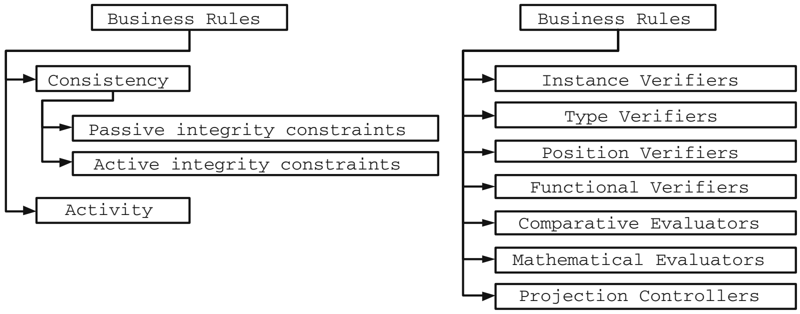


Fig. 3. Business rules classification schemas according to Herbst and Ross

Table 1. Mapping of rules from Ross and Herbst schemas to ECA paradigm

Ross	Herbst	ECA
Instance Verifiers	Passive Constraints	$event = state\_change\_of\_anchor$ $condition = \{rule\_body \mid TRUE\}$ $action = \{err\_msg \mid err\_data\_elimination\}$
Type Verifiers		
Position Verifiers		
Functional Verifiers		
Comparative Evaluators		
Mathematical Evaluators		
Projection Controllers	Activity	$event = state\_change\_of\_anchor$ $condition = \{rule\_body \mid TRUE\}$ $action = rule\_action$
	Active Constraints	

data type (or rule) for whose instances a rule is specified. Any state changes on the anchor would constitute the firing event. Thus for each rule its event can be made explicit as data (anchor) state changes.

- Each rule has a condition. If the rule has condition defined on business data, then the condition is explicit. If the condition is blank, then the default condition can be used which always evaluates to $TRUE$. Thus for each rule its condition can be made explicit – as business-defined or default one.
- Each rule has an action – explicit or implicit. If action is specified explicitly (for projection controllers in Ross method and activity rules in Herbst schema), then it is executed when the rule is fired. Otherwise action is implicit. Ross defines first six families of rules from his method as *rejectors* – i.e. rules do not allow updates of data which violate rules. However, if constraint is violated then either user must be informed and take corrective actions or data which violates the constraint must automatically be eliminated. Thus two special action types might be assigned to each rule which is a constraint: issuing error message or automatically eliminating erroneous data. These types of actions do have the business meaning and are important because it is the business user who must define how and what corrective actions will be carried out. Therefore these types of actions also must be explicitly present in the model. Thus for each rule its action can be made explicit.

Table 1 shows how the mapping is accomplished from Ross and Herbst et al. classification schemas to the ECA paradigm.

In Figure 4 the rule *The organization unit must have at least three employees* is presented. The rule is of passive constraint type according to Herbst and of instance verifiers type according to Ross.

The rule is mapped to ECA schema the following way:

- event is implicit – the rule must be fired when the number of employees assigned to a specific unit is changed. The alternatives are: an employee is transferred to another unit or leaves the organization. Both cases change

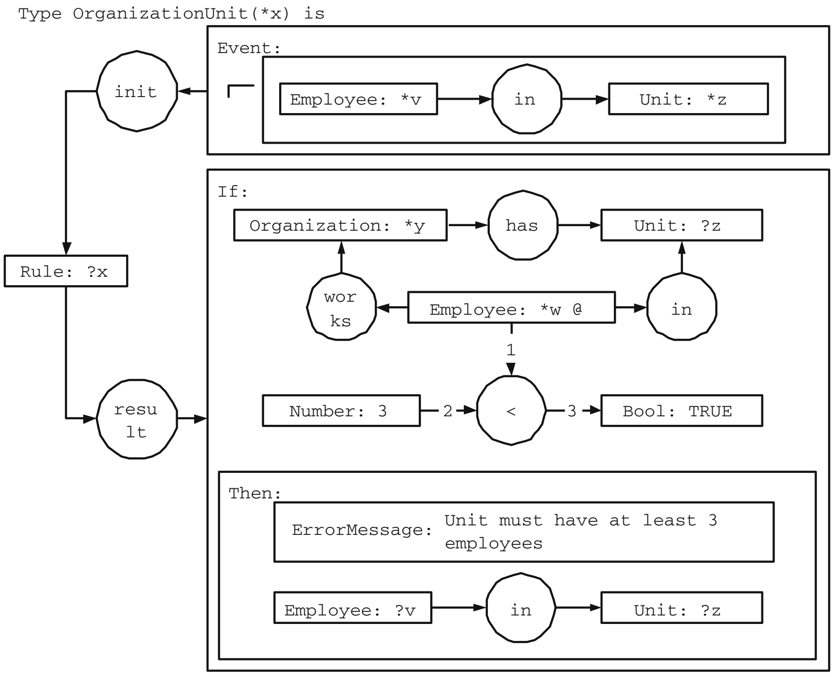


Fig. 4. Business rule *The organization unit must have at least three employees* in CG graphical notation

the relationship *employee-unit* thus the event is defined as deletion of the relationship between the unit and any of employees assigned to the unit.
- condition is explicit – number of employees must be at least 3.
- action is implicit – thus the error message is defined and the relationship is restored. It is posible to define another corrective actions, such as delete the unit itself and reassign the remaining employees to the parent unit. But this is to be decided by the rule owners. In Figure 4 the first alternative is presented.

In Figure 5 the rule *The same course can be scheduled again only after 90 days* is presented. The rule is of passive constraint type according to Herbst and of functional verifiers type according to Ross.

The rule is mapped to ECA schema the following way:

- event is explicit – the rule must be fired upon the attempt to schedule the course.
- condition is explicit – the time period between the last time the course was held is at least 90 days.
- action is implicit – thus the error message is defined and the new date for the course is calculated. It is posible to define another corrective actions,

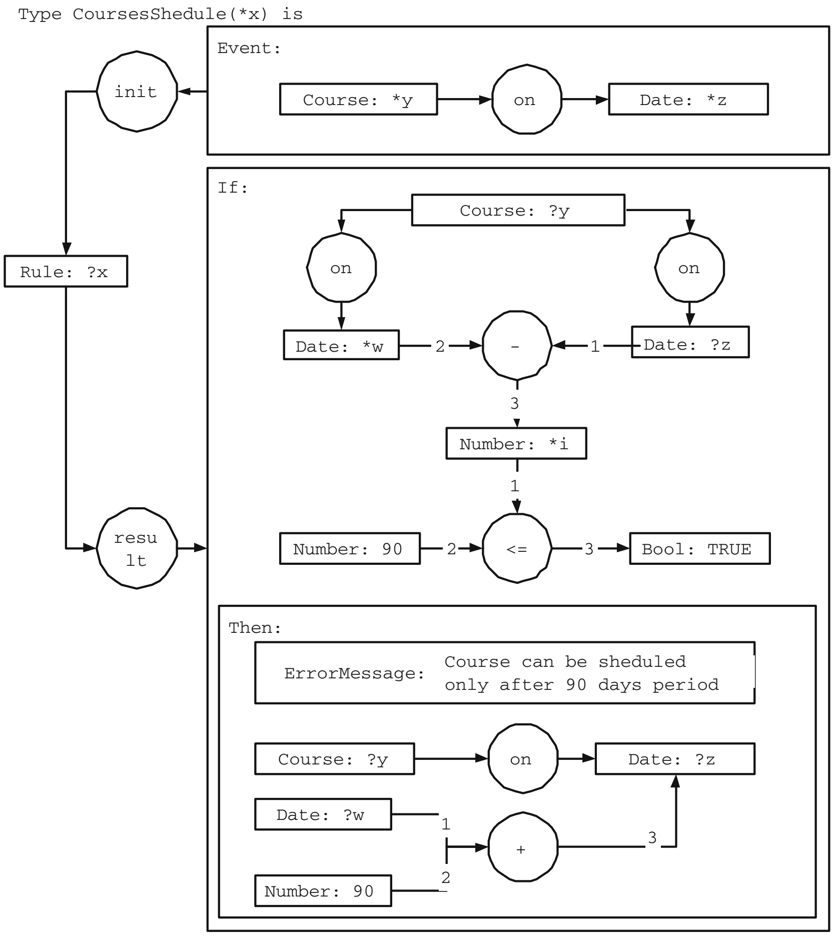


Fig. 5. Business rule *The same course can be scheduled again only after 90 days* in CG graphical notation

such as delete the course from the schedule or leave the date, but change the course. But again this is to be decided by the rule owners. In Figure 5 the first alternative is presented.

3.4 Business Rules Implementation

Developing business rules model can be regarded as an independent activity with a purpose to have and maintain the consistent business knowledge repository (as it was done in [9]). However, the implementation of such a model is the next important step in business IS development. In [13] it was proposed to implement business rules using active databases triggers technology because of the semantic similarity of business rule and ECA rule. A basic set of trigger automatic generation rules was presented where rules were modeled using concept of demon. Having all business rules modeled in the CG knowledge base augmented with rule base element gives additional advantages:

1. All business rules may be directly mapped to trigger parts (trigger clause, condition, and body) because the defined type *Rule* addresses rule event, condition, and action explicitly. In automatic generation engine the mapping from model to implementation is direct thus yielding simpler mapping rules set.
2. Concept types hierarchy and relation types hierrachy constitute the ontology of business domain which may be used for merging or diverging different models from the similar domains.

Having business rules model in CG it is possible to define the set of trigger generation rules from rule base and implement an automatic generation process which would translate rule base to a set of triggers for a specific active database management system because CG have a standard interchange format and the type *Rule* enforces the ECA-based rule structure.

4 Conclusions and Future Research

In this paper we argue that trends in business systems towards explicit knowledge management force business IS development research focus on business knowledge representation and implementation techniques. Business rules approach for business knowledge structuring is convenient for both parties - business people and system analysts - because business rule is the commonly used and well understood concept.

The necessity of the explicit and separate business rules model yields the question what modeling language is the most appropriate. Before making attempts to motivate the choice of a modeling language we define the requirements for such a language: it must be formal, have graphical notation, support standard interchange format and be easily comprehensible for novice users. Conceptual graphs satisfy all these requirements therefore we advocate its usage for business rules modeling.

To facilitate the business rules modeling using conceptual graphs we propose to add one more element to the CG knowledge base framework defined in conceptual graphs by Sowa. The new element rule base is defined as a set of conceptual graphs which represent business rules. Such framework allows to address business rules explicitly thus satisfying the primary requirement of business rules systems. Additionaly we define the business rule metamodel and require that all conceptual graphs in rule base must be of the type *Rule*.

The business rule metamodel as defined in this paper is applicable for all business rules despite their diversity. To prove that fact we analyse and demonstrate that Ross and Herbst classification schemas smoothly map to the ECA paradigm.

Having business rules model represented using CG rule base it is possible to achieve automatic business rules implementation because type *Rule* is based on the ECA paradigm which directly translates to active databases trigger. For this purpose a formerly proposed set of translation rules to active databases trigger must be refined with regard to the rule base element.

References

1. ACT-NET Consortium: The Active Database Management Systems Manifesto: A Rulebase of ADBMS Features. ACM Sigmod Record, Vol. 25(30) (1996) 40–49
2. Badawy, M., Richta, K.: Deriving Triggers from UML/OCL Specification. In: Kirikova M. (eds.): Information Systems Development: Advances in Methodologies, Components and Management. Kluwer Academic/Plenum Publishers (2002)
3. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley (2000)
4. Boyd, L.: CDM RuleFrame – The Business Rule Implementation Framework That Saves You Work. ODTUG 2001 conference proceedings (2001)
5. Chein, M., Mugnier, M.L.: Conceptual Graphs: Fundamental Notions. Revue d'Intelligence Artificielle, Vol. 6, No.4 (1992) 365–406
6. Demuth, B., Hussmann, H., Loecher, S.: OCL as a Specification Language for Business Rules in Database Applications. In: Gogolla, M., Kobryn, C. (eds.): UML 2001. LNCS, V. 2185. Springer-Verlag Berlin (2001) 104–117
7. Kolber, A., Hay, D., Healy, K.A., Hall, J. et al.: GUIDE Business Rules Project: Final Report. GUIDE International Corporation (1997)
8. Herbst. H., Knolmayer, G., Myrach, T., Schlesinger, M.: The Specification of Business Rules: A Comparison of Selected Methodologies. In: Verijn-Stuart, A.A., Olle T.-W. (eds.): Methods and Associated Tools for the Information System Life Cycle. Elsevier Amsterdam (1994) 29–46
9. Gerbe, O., Keller, R.K., Mineau, G.W.: Conceptual Graphs for Representing Business Processes in Corporate Memories. In: Mugnier, M.-L., Chein, M. (eds.): ICCS'98, Lecture Notes in Artificial Intelligence, Vol.1453. Springer-Verlag Berlin Heidelberg New York (1998) 401–415
10. Ross, R.G.: The Business Rule Book. Classifying, Defining and Modeling Rules. Business Rules Solutions, LLC, Houston, Texas (1997)
11. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley Publishing Company Reading (1984)
12. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks/Cole, Pasific Grove et al. (2000)
13. Valatkaite, I., Vasilecas, O.: Application Domain Knowledge Modeling Using Conceptual Graphs. In: Kirikova, M. (eds.): Information Systems Development: Advances in Methodologies, Components and Management. Kluwer Academic/Plenum Publishers (2002) 193–202
14. Warmer, J., Kleppe, A.: The Object Constraint Language. Precise modeling with UML. Addison Wesley Longman (1999)

SnoopIB: Interval-Based Event Specification and Detection for Active Databases*

Raman Adaikkalavan and Sharma Chakravarthy

Computer Science and Engineering Department
The University of Texas at Arlington
Arlington, TX 76019, USA
{adaikkal,sharma}@cse.uta.edu

Abstract. Snoop is an event specification language developed for expressing primitive and composite events that are part of Event-Condition-Action (or ECA) rules. In Snoop, an event was defined to be an instantaneous, atomic (happens completely or not at all) occurrence of interest and the time of occurrence of the last event in an event expression was used as the time of occurrence for the entire event expression. The above detection-based semantics does not recognize multiple compositions of some operators – especially Sequence – in the intended way. In order to recognize all event operators, in all contexts, in the intended way, operator semantics need to include start time as well as end time for an event expression (i.e., interval-based semantics). In this paper, we formalize Snoop Interval-Based (SnoopIB), the occurrence of Snoop event operators and expressions using interval-based semantics. The algorithms for the detection of events using interval-based semantics introduce some challenges, as not all the events are known (especially their starting points).

1 Introduction

There is consensus in the database community on the Event-Condition-Action rules (or ECA) as being one of the most general formats for expressing rules in an active database management system (ADBMS). As the event component was the least understood (conditions correspond to queries, and actions correspond to transactions) part of the ECA rule, there is a large amount of work that is done on the language for event specification. Snoop [1, 2] was developed as the event specification component of the ECA rule formalism used as a part of the Sentinel project [3-6] on active object-oriented DBMS. Snoop supports expressive ECA rules that include coupling modes and parameter (or event consumption) contexts.

An event is an indicator of happening, which can be either primitive (e.g., depositing cash in bank) or composite (e.g., depositing cash in bank, followed by withdrawal of cash from bank). Primitive events occur at a point in time (i.e., time of depositing). Composite events occur over an interval (i.e., interval starts at the time cash is deposited and ends when cash is withdrawn). Thus, primitive events are detected at a point in time, whereas the composite events can be detected either at the

* We would like to acknowledge the support of AF/Spawar grant (F 30602-02-2-0134) and the NSF grant (IIS-0121297 and IIS-0123730) for this work

end of the interval (i.e., detection-based semantics, where start of the interval is not considered) or can be detected over the interval (i.e., interval-based semantics).

In all event specification languages used in Active DBMSs (Snoop [1, 2], COMPOSE [8, 9], Samos [10, 11], ADAM [12, 13], ACOOD [14, 15], event-based conditions [16], and Reach [17-19]), events are considered as "instantaneous", although an event occurs over an "interval". Because of this, all these event specification languages detect a composite event at the *end of an interval* over which it occurs (i.e., detection-based semantics). When events are detected using the detection-based semantics, where *event occurrence* and *event detection* is not differentiated, it leads to some unintended semantics as pointed out in [7, 25] when certain operators, such as sequence, are composed more than once. Interval-based semantics is used to overcome the problems that are caused by the detection-based semantics and is explained in the following subsections with some intuitive examples.

Subsections 1.1 – 1.2 explains event detection, detection-based semantics, and highlights how the interval-based semantics is used in event detection and why the distinction between detection-based and interval-based semantics is important with a detailed example. The reader is referred to [7] [25] for a good description of the differences between the AI and database approaches.

1.1 Event Detection

In this section we will see the conditions that are needed to detect an event. Primitive events are predefined in the system and are detected at the time of occurrence. Composite event detection on the other hand involves two steps and they are 1) checking the detection condition based on the operator semantics, and 2) time of detection. These are the two main steps and they are handled differently in detection-based semantics and interval-based semantics and are explained below.

Let us take an example where a stock trading agent wants to indicate a stock buyer, *"When Dow Jones Industrial Average (DJIA) increases by 5% followed by a 5% price increase in Sun Microsystems shares and 2% price increase in IBM shares"*. Stock trading agent uses Snoop event operators to express this requirement. "If **(DJIA5 ; (Sun5 $\wedge$ IBM2))** then indicate the buyer". DJIA5, Sun5, IBM2 are primitive events that correspond to *DJIA increases by 5%*, *5% price increase in Sun shares, 2% price increase in IBM shares* respectively. (Sun5 $\wedge$ IBM2) and (DJIA5 ; (Sun5 $\wedge$ IBM2)) are composite events, where ";" (snoop "sequence" event operator) represents *followed by* condition and "$\wedge$" (snoop "and" event operator) represents *And* condition. Both these operators are binary operators. ";" detects sequence of two events, when ever the first event happens before the second event. "$\wedge$" detects a "And" event when ever both the events happen. Semantics for these operators are different for detection-based semantics and interval-based semantics and are explained in the following subsections.

1.1.1 Detection-Based Semantics

Let us assume that primitive events DJIA5, Sun5 and IBM2 occur at *10.30 a.m.*, 10 a.m., and 11 a.m. respectively. Fig. 1 depicts events that are detected, along with the time of occurrence and detection.

1. Primitive events are detected at the time of occurrence, thus events DJIA5, Sun5 and IBM2 are detected at 10.30 a.m., 10 a.m., and 11 a.m. respectively.
2. Composite event (Sun5 $\wedge$ IBM2) detection involves two steps as previously mentioned.
   a. "And" condition is satisfied since both the events have occurred.
   b. Since events are considered as "instantaneous" and are detected at end of the interval, composite event (Sun5 $\wedge$ IBM2) is detected at 11 a.m. (though it occurred over an interval from 10 a.m. to 11 a.m.)
3. Similarly composite event (DJIA5 ; (Sun5 $\wedge$ IBM2)) detection involves two steps and they are
   a. "Followed by" condition is satisfied when event DJIA5 happens before event (Sun5 $\wedge$ IBM2). In our example, this condition is satisfied, since event DJIA5 is detected at 10.30 a.m. in step 1, event (Sun5 $\wedge$ IBM2) is detected at 11 a.m. in step 2.b and thus primitive event DJIA5 *happens before the* event (Sun5 $\wedge$ IBM2) (i.e., 10.30 a.m. < 11 a.m.).
   b. Composite event (DJIA5 ; (Sun5 $\wedge$ IBM2)) is detected at 11 a.m. (even though it occurs over an interval from 10.30 a.m. to 11 a.m. and the buyer is indicated.)

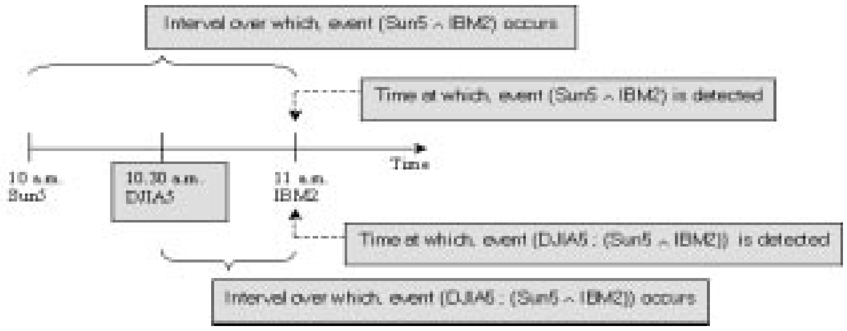

Interval over which, event (Sun5 ∧ IBM2) occurs
Time at which, event (Sun5 ∧ IBM2) is detected
Time
10 a.m.
Sun5
10.30 a.m.
DJIA5
11 a.m.
IBM2
Time at which, event (DJIA5 ; (Sun5 ∧ IBM2)) is detected
Interval over which, event (DJIA5 ; (Sun5 ∧ IBM2)) occurs

Fig. 1. Detection-Based Semantics Example

In this example, the buyer is indicated, *which is unintended*, since primitive event Sun5 had occurred well before primitive event DJIA5. But this happens, because the condition checking in step *3.a. fails* to capture the correct semantics, since it does not consider the start of interval.

From this, it is discernible that detection-based semantics does not detect events in the intended way. The detection-based semantics typically used by all the aforementioned event specification languages used in Active DBMSs do not differentiate between *event occurrence* and *event detection*. As we have seen, with detection-based semantics the buyer is indicated in both the cases, even though the primitive event Sun5 had occurred well before primitive event DJIA5 in the second case, which is unintended.

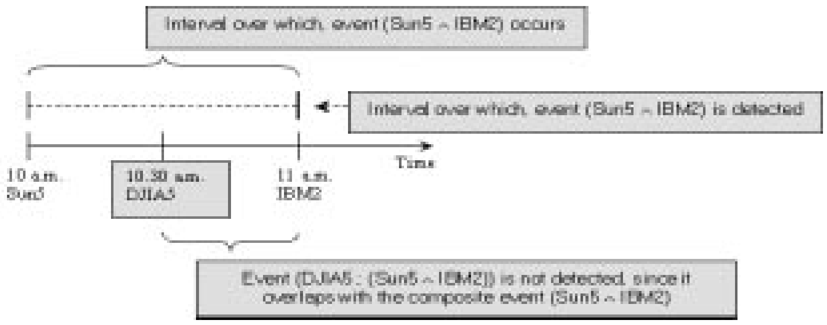

Interval over which, event (Sun5 ∧ IBM2) occurs
Interval over which, event (Sun5 ∧ IBM2) is detected
Time
10 a.m. Sun5
10.30 a.m. DJIA5
11 a.m. IBM2
Event (DJIA5 ; (Sun5 ∧ IBM2)) is not detected, since it overlaps with the composite event (Sun5 ∧ IBM2)

Fig. 2. Interval-Based semantics Example

1.1.2 Interval-Based Semantics

Examples that were used to explain the detection-based semantics are used to explain interval-based semantics in this section. Interval-based semantics, where both start and end interval is considered is based on the fact that real life events have an interval and are not instantaneous.

Let us assume that primitive events DJIA5, Sun5 and IBM2 occur at *10.30 a.m.*, 10 a.m., and 11 a.m. respectively. Fig. 2 depicts events that are detected, along with the time of occurrence and detection.

1. Primitive events are detected at the time of occurrence, thus events DJIA5, Sun5 and IBM2 are detected at 10.30 a.m., 10 a.m., and 11 a.m. respectively.
2. Composite event (Sun5 $\wedge$ IBM2) detection involves two steps as previously mentioned.
   a. "And" condition is satisfied since both the events have occurred.
   b. Since start of the events is considered, events are detected over the interval. Composite event (Sun5 $\wedge$ IBM2) is detected over the interval [10 a.m., 11 a.m.]
3. Similarly composite event (DJIA5 ; (Sun5 $\wedge$ IBM2)) detection involves two steps and they are
   a. "Followed by" condition is *not satisfied*, since event DJIA5 does not happen before event (Sun5 $\wedge$ IBM2). In our example, this condition is not satisfied, since the primitive event DJIA5 is detected at 10.30 a.m. in step 1, composite event (Sun5 $\wedge$ IBM2) is detected over the interval [10.00 a.m., 11 a.m.] in step 2.b and thus primitive event DJIA5 *does not happens before the* event (Sun5 $\wedge$ IBM2) (i.e., 10.30 a.m. $\not<$ 10 a.m.).
4. Composite event (DJIA5 ; (Sun5 $\wedge$ IBM2)) *is not detected* and the buyer is not indicated, which is the intended event detection.

1.2 Our Contribution

Interval-based semantics, where the start interval as well as the end interval of an event is considered, is not just another way to detect events, but an essential one to ascertain the correct event detection. With the interval-based semantics for the event operators in ECA rules all the events are detected in the intended way. Galton [7] [25] has pointed out this discrepancy between database work where detection semantics has been used to define the operators in contrast to work in AI where occurrence has played a dominant role for inference than detection and hence interval semantics has been used [20, 21].

In this paper, we present interval-based semantics for all Snoop operators for *recent context* drawing upon the approach presented in [7] [25] for the general or unrestricted context. We have also developed algorithms for event detection using interval-based semantics for different contexts including the general or unrestricted context and are implemented using event graphs, please refer [29] for more details.

Outline. The rest of the paper is organized as follows. Section 2 explains related work. Section 3 explains the interval-based semantics of Snoop in the unrestricted context. Section 4 extends the above to recent context. Section 5 has conclusions and future work.

2 Related Work

[26] explains why the interval-based semantics is needed for event detection with some *concrete* examples, using Snoop operators, but does not deal with formal semantics, algorithms and implementation for any of the context in Snoop. There has been a considerable amount of work done in the interval-based semantics. [27] explains the event detection using the duration-based (i.e., interval-based) semantics, but why it is needed, what operators are supported, how it is implemented and the formal semantics is not explained.

Snoop [1,2] uses event graphs to detect the composite event, whereas Samos [10,11] uses Petri nets to detect the composite events, likewise all the aforementioned event specification languages detects the composite event using different approaches, but all of them use detection-based semantics, which has some problems as we have seen before. Details of event detection by other event specification languages and why they are not sufficient can be found in [29].

Algorithms for event composition and event consumption, which make use of accuracy interval based time stamping is illustrated along with a window mechanism to deal with varying transmission delays when composing events from different sources, are dealt in [28]. The paper claims that event consumption modes like recent and chronicle can be unambiguously defined by using an accuracy interval order that guarantees the property of time consistent order. Even though this system uses “accuracy interval based time stamping” guaranteeing the time consistent order for the event arrival, it uses the detection-based semantics for the composite event detection, which has the same drawbacks.

3 Interval-Based Semantics of Snoop

For the purpose of this paper, we assume an equidistant discrete time domain having "0" as the origin and each time point represented by a non-negative integer as shown in Fig. 3. The granularity of the domain is assumed to be specific to the domain of interest. A primitive event is *detected* atomically at a point on the time line. In object-oriented databases, interest in events comes from the state changes produced by method executions by an object. Similarly, in relational databases, interest in events comes from the data manipulation operations such as insert, delete, and update. Similar to these database (or domain specific) events there can also be temporal events that are based on time or explicit events that are detected by an application program (outside of a DBMS) along with its parameters. Detection-based semantics was adopted as begin and end events were of significance in most of the database related work.

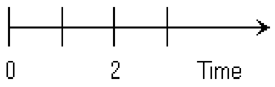


Fig. 3. Time Line

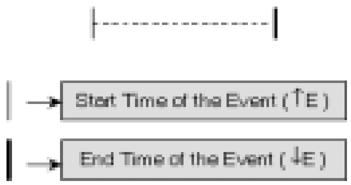


Fig. 4. Event Notations

3.1 Primitive Events

Primitive events are a finite set of events that are pre-defined in the (application) domain of interest. Primitive events are distinguished as domain specific, temporal and explicit events (for more detail refer to [1, 2, 22]). For example, *a method execution* by an object in an object-oriented database is a primitive event. These *method executions* can be grouped into *before* and *after* events (or *event types)* based on where they are detected (*immediately before* or *after the method call*). An event occurs over a time interval and is denoted by E [t_1, t_2] (where E is the event, t_1 is the start interval of the event denoted by ↑E, t_2 is the end interval of the event denoted by E↓). In the case of primitive events, the start and the end interval are assumed to be the same (i.e., $t_1 = t_2$). For events that span over an interval, the event *occurs* over the interval [t_1, t_2] and *detected* at the end of the interval.

3.2 Event Expressions

For many applications, supporting only primitive events is not adequate. In many real-life applications, there is a need for specifying more complex patterns of events such as, arrival of a report followed by a detection of a specified object in a specific area. They cannot be expressed with a language that does not support expressive event operators along with their semantics. An appropriate set of operators along with the closure property allows one to construct complex composite events by combining primitive events and composite events in ways meaningful to an application interested in situation monitoring. To facilitate this, we have defined a set of event operators along with their semantics. Snoop [1, 2] is an event specification language that is used to specify combinations of events using Snoop operators such as And, Or, Sequence, Not, Aperiodic, Periodic, Cumulative Aperiodic, Cumulative Periodic, and PLUS. Motivation for the choice of these operators and how they compare with other event specification languages can be found in [1, 2].

3.3 Composite Events

Composite events are constructed using primitive events and event operators in a recursive manner. A composite event consists of a number of primitive events and operators; and the set of primitive events of a composite event are termed as constituent events of that composite event. A composite event is said to occur *over an interval*, but is *detected* at the point when the last constituent event of that composite event is detected. The detection and occurrence semantics is clearly differentiated and the detection is defined in terms of occurrence as shown in [7] [25]. Note that occurrence of events cannot be defined in terms of detection which was the problem with the earlier detection-based approaches.

We introduce the notion of an *initiator*, *detector*, and *terminator* for defining event occurrences. A composite event occurrence is based on the initiator, detector and terminator of that event which in turn are constituent events of that composite event. An *initiator* of a composite event is the first constituent event whose occurrence starts the composite event. *Detector* of a composite event is the constituent event whose occurrence detects the composite event, and *terminator* of a composite event is the constituent event that is responsible for terminating the composite event. For some operators, the detector and terminator are different (e.g., Aperiodic). For many operators, the detector and terminator are the same (e.g., Sequence).

A Composite event E occurs *over a time interval* and is defined by E $[t_1, t_2]$ where E is a composite event, t_1 is the start time of the composite event occurrence and t_2 is the end time of composite event occurrence (t_1 is the starting time of the first constituent event that occurs (*initiator*) and t_2 is the end time of the detecting or terminating constituent event (*detector or terminator*) and they are denoted by $\uparrow$E and E$\downarrow$ respectively).

End of an event: $O(E\downarrow, t) \triangleq \exists t' \leq t\ (O(E, [t', t]))$.

Start of an event: $O(\uparrow E, t) \triangleq \exists t'\ (t \leq t' \wedge O(E, [t, t']))$.

3.3.1 Overlapping Event Combinations

Another aspect of event occurrences of the constituent events of a composite event is that they can be either overlapping or disjoint. When the events are allowed to overlap, all combinations in which two events can occur [20] are shown in Fig. 5. All the operators formally defined in this paper assume that events occur in an overlapping fashion.

3.3.2 Semantics of Event Operators

The formal definitions for unrestricted (general) contexts are provided in [7] [25] except for the *Plus operator*.

Formal Definition for Plus Event in Unrestricted Context. E = O (Plus (E_1, n) [t, t]). E is the Plus event, E_1 is the initiator, n is the terminator. A Plus operator is used to specify a relative time event [23]. A Plus operator combines two events E_1 and E_2 where E_1 can be any type of event and E_2 is a time string [t]. The Plus event occurs after time [t], after the event E_1 occurs. The formal definition for the Plus operator for the unrestricted context is shown below.

$O (\text{Plus} (E_1, n) [t, t]) \triangleq \exists t'<t (O (E_1\downarrow, t') \wedge t = t' + n)$.

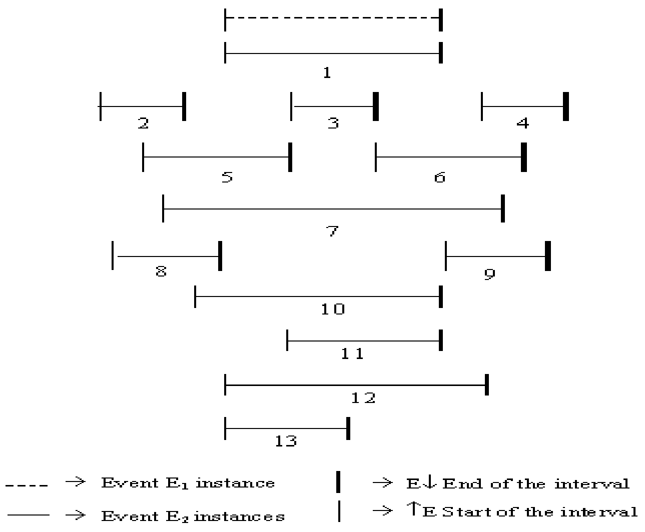

1
2
3
4
5
6
7
8
9
10
11
12
13
---- -> Event E₁ instance
—— -> Event E₂ instances
-> E↓ End of the interval
-> ↑E Start of the interval

Fig. 5. Overlapping event combinations

4 Parameter Contexts

A large number of events are generated when unrestricted context is used. When we studied many application domains, it turned out that these application domains may not be interested in the unrestricted context all the time but need mechanisms to tailor the semantics of event expression to their domain needs. In order to provide more meaningful event occurrences to match application needs, Snoop introduced several parameter contexts [1, 2]: Recent, Chronicle, Continuous, and Cumulative. These parameter contexts filter the events (or the history) generated by the unrestricted context. We briefly describe below the motivation for the introduction of recent context which generates a subset of events generated by the unrestricted context.

Recent Context: In applications where events are happening at a fast rate and multiple occurrences of the same event only refine the previous value can use this context.

Only the most recent or the latest initiator for any event that has started the detection of a composite event is used in this context. This entails that the most recent occurrence just updates (summarizes) the previous occurrence(s) of the same event type. In this context, *not all occurrences* of a constituent event will be used in the composite event detection. An initiator will continue to initiate new event occurrences until a *new initiator* or a *terminator* occurs.

4.1 Operator Semantics in Recent Context

In this section, we extend the formal semantics to recent context. In addition, we provide some algorithmic and implementation details with respect to the event detection in recent context. For a full description, please refer to [24][29]. Below, "O" represents the occurrence-based Snoop semantics.

4.2 Event Histories

The above intuitive explanations of contexts are based on the event occurrences over a time line. In this section, using the notion of *event histories,* we formalize these definitions to take the parameter contexts into account. An event history maintains a history of event occurrences up to a given point in time. Suppose e_1 is an event instance of type E_1 then E_1 [H] represents the event history that stores all the instances of the event E_1 (namely e^i_1). In order to extend these definitions to parameter contexts the following notations are used.

E_i [H] => Event history for event e_i,

t_{si} – Starting time of an event e_i, t_{ei} – Ending time of an event e_i

Below, we first describe the history-based event occurrences intuitively before defining them formally.

4.2.1 Occurrence Semantics in Recent Context

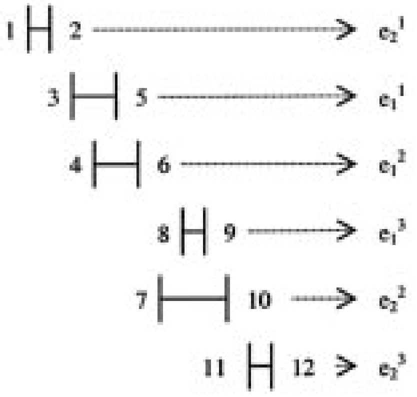
Sequence operator in Unrestricted Context: Event histories can be used for the detection of the ";" operator defined in Section 3.3. With event histories E_1 [H] = {(3, 5), (4, 6), (8, 9)}, E_2 [H] = {(1, 2), (7, 10), (11, 12), for events shown in Fig. 6, (E_1; E_2) generates the following pairs of events in the unrestricted context: {(e_1^1, e_2^2) [3,10], (e_1^2, e_2^2) [4, 10], (e_1^1, e_2^3) [3,12], (e_1^2, e_2^3) [4, 12], (e_1^3, e_2^3) [8, 12]}.

Sequence operator in Recent Context: The ";" operator in recent context is formally defined as follows:

$$\begin{aligned} O(E_1; E_2, [t_{s1}, t_{e2}]) \triangleq \{&\forall E_1 \in E_1 [H] \wedge \forall E_2 \in E_2 [H] \wedge \\ &\{(O(E_1, [t_{s1}, t_{e1}]) \wedge O(E_2, [t_{s2}, t_{e2}]) \wedge (t_{s1} \le t_{e1} < t_{s2} \le t_{e2}))\} \\ &\wedge (\nexists E_1' [t_{start}, t_{end}] \mid (t_{e1} < t_{end} \le t_{e2}) \wedge E_1' \in E_1 [H])\} \end{aligned}$$

In order to formally define the Sequence event in the recent context, take an event pair E_1 and E_2 from the event histories E_1 [H] and E_2 [H] respectively. For this event pair to be a Sequence event in the recent context, there should not be an occurrence of any other instance of event E_1 from the event history E_1 [H] in the interval formed by this event pair.

Formalization of the sequence operator in the recent context is explained below using the event histories E_1 [H] = {(3, 5), (4, 6), (8, 9)}, E_2 [H] = {(1, 2), (7, 10), (11, 12)} shown in Fig. 6. In this context only the most recent initiator is used (see section 4). In the above example, when the event e_2^1 occurs over the interval [1,2] there is no event in the event history of E_1 that satisfies the ";" operator condition. Event e_2^2 occurs over the interval [7, 10]. It is not paired with event e_1^1 because there is an occurrence of event e_1^2 [4, 6] in the interval formed by the end time of event e_1^1 [3, 5] and start time of event e_2^2 [7, 10], which does not satisfy the condition given above. Event e_2^2 does not detect the sequence event in the recent context with the event e_1^2 since event e_1^3 is the recent initiator. Event e_1^3 cannot pair with the event e_2^2 since it does not satisfy the ";" semantics. Similarly, when the event e_2^3 occurs it detects the recent event with the event pair (e_1^3, e_2^3) over the interval [8, 12]. Thus, events in recent context are: {(e_1^3, e_2^3) [8, 12]}.



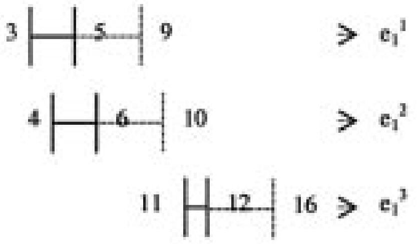


Fig. 6. Examples for Sequence Operator **Fig. 7.** Examples for Plus Operator

Plus Operator in Unrestricted Context: Plus event occurs only once after the time interval specified by 'n' after the event E_1 occurs and denoted by (Plus (E_1, n) [t, t]). By the definition of the Plus event the start time and end time are the same. For example, Plus event (Plus (E_1, 4)) is taken which is detected after 4 units after the occurrence of event E_1 to explain the unrestricted and recent context. For the events shown in Fig. 8 "Plus" event defined in Section 3.3 generates the following pairs of events in the unrestricted context: {(e_1^1, 4) [9, 9], (e_1^2, 4) [10, 10], (e_1^3, 4) [16, 16]}.

Plus Operator in Recent Context: Below, in the formal definition for the Plus event in the recent context, event E_1 is assumed to end at a time point [t'] and Plus event at the time point [t]. Plus event occurs in the recent context when ever there is no other instance of E_1 event from E_1 [H] occurs in the interval formed by [t'] and [t]. This is given as the condition ($\nexists$ (E_1'$\downarrow$, t'') | (t' < t'' ≤ t)).

$$O\ (\text{Plus}\ ((E_1, n), [t, t])) \triangleq \exists t' < t\ (O\ (E_1\downarrow, t') \wedge t = t' + n \wedge (\nexists\ (E_1'\downarrow, t'')\ |\ (t' < t'' \leq t))).$$

Formal definition given above is explained with the events shown Fig. 7. Event e_1^1 initiates a Plus event at the time point [5]. When the event e_1^2 occurs it initiates a new

Plus event and terminates the Plus event that was initiated previously. At the time point [10] Plus event is detected in the recent context. Similarly event e_1^3 detects a Plus event at the time point [16]. Events that are detected in recent context: {(e_1^2, 4) [10, 10], (e_1^3, 4) [16, 16]}.

NOT Operator in Unrestricted Context: "¬" Operator can be expressed as the Sequence of E_1 and E_2 where there is no occurrence of the event E_3 in the interval formed by these events. Thus, explanation of the "¬" Operator definition is same as the ";" operator with one additional condition. This condition stipulates that there cannot be an occurrence of the event E_3 from E_3 [H] in the interval formed by the end time of the event E_1 and the start time of the event E_2. Event histories can be used for the detection of the NOT operator ¬ (E_3) [E_1, E_2], defined in the section 3.3. With event histories E_1 [H] = {(3, 5), (4, 6), (8, 9)}, E_2 [H] = {(1, 2), (7, 10), (11, 12)}, E_3 [H] = {(5, 5)} shown in Fig. 8 "¬" event generates the following pair of events {(e_1^2, e_2^2) [4, 10], (e_1^2, e_2^3) [4, 12], (e_1^3, e_2^3) [8, 12]}.

Not Operator in Recent Context: The "¬" operator in recent context is formally defined as follows:

$$O (\neg (E_3) [E_1, E_2], [t_{s1}, t_{e2}]) \triangleq \{\forall E_1 \in E_1 [H] \wedge \forall E_2 \in E_2 [H] \wedge \forall E_3 \in E_3 [H] \wedge$$
$$\{(O (E_1, [t_{s1,} t_{e1}]) \wedge O (E_2, [t_{s2}, t_{e2}]) \wedge (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2}) \wedge \neg O_{in} (E_3, [t_{e1}, t_{s2}]))\}$$
$$\wedge (\nexists E_1' [t_{start,} t_{end}] \mid (t_{e1} < t_{end} \leq t_{e2}) \wedge E_1' \in E_1 [H])\}.$$

Formally, for an event pair E_1 and E_2 to be in the recent context, there cannot e an occurrence of any other instance of event E_1 from the event history E_1 [H] between this pair. Formalization of the "¬" operator in the recent context is explained below using event histories E_1 [H] = {(3, 5), (4, 6), (8, 9)}, E_2 [H] = {(1, 2), (7, 10), (11, 12)}, E_3 [H] = {(5, 5)} shown in Fig. 8. Occurrence of event e_2^1 does not detect any event since the event history E_1 [H] is empty. Event e_1^1 initiates "¬" event in the recent context, and is terminated by event e_3^1. "¬" Event in the recent context is initiated by event e_1^2. This event is terminated by the occurrence of the event e_1^3, which acts as the most recent initiator. Event occurrence e_2^2 does not pair with the initiator e_1^3 since it does not satisfy the condition ($t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2}$), where as the event e_2^3 detects a recent event with the initiator e_1^3 since there are no other instances of event E_1 occurrence in the interval formed by this event pair and this satisfies both the conditions ($t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2}$) and ($\nexists E_1' [t_{start,} t_{end}] \mid (t_{e1} < t_{end} \leq t_{e2}) \wedge E_1' \in E_1 [H]$). Thus events detected by "¬" in recent context are: {(e_1^3, e_2^3) [8, 12]}.

OR Event: The semantics of "∇" does not change with the context as each occurrence is detected individually.

Aperiodic Operator in Unrestricted Context: Detection of (A (E_1, E_2, E_3), [t_{s1}, t_{e1}]) in the unrestricted context defined in Section 3.3 using event histories E_1 [H] = {(3, 5), (4, 6)}, E_2 [H] = {(1, 2), (8, 9), (7, 10), (11, 12)}, E_3 [H] = {(11, 11)} shown in Fig. 9 detects the following pair of events {(e_1^1, e_2^2) [8, 9], (e_1^2, e_2^2) [8, 9], (e_1^1, e_2^3) [7, 10], (e_1^2, e_2^3) [7, 10]}.

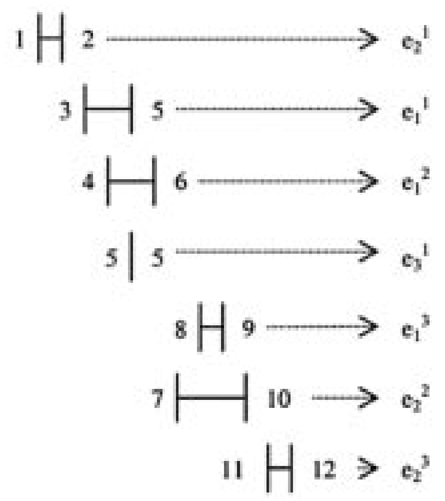


Fig. 8. Examples for Not Operator

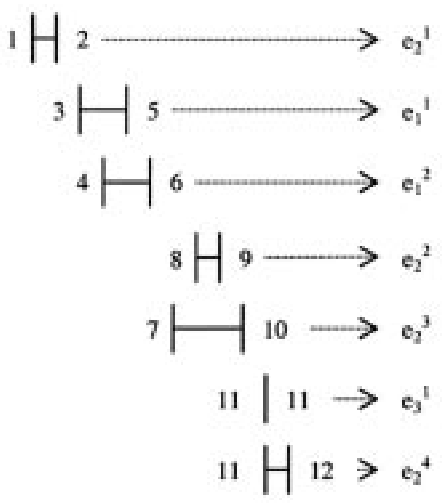


Fig. 9. Examples for A and A* Operators

Aperiodic Operator in Recent Context: The "A" operator in recent context is formally defined as follows:

$$O\,(A\,(E_1, E_2, E_3), [t_{s1}, t_{e1}]) \triangleq \{\forall E_1 \in E_1\,[H] \wedge \forall E_2 \in E_2\,[H] \wedge \forall E_3 \in E_3\,[H] \wedge$$
$$\{O\,(E_2, [t_{s1}, t_{e1}]) \wedge \exists t < t_{s1}\,(O\,(E_1{\downarrow}, t) \wedge \neg O_{in}\,(E_3, [t+1, t_{e1}]))\}$$
$$\wedge\,(\nexists\, E_1'\,[t_{start,}\, t_{end}]\,|\,(t < t_{end} \le t_{e1}) \wedge E_1' \in E_1\,[H])\}.$$

Aperiodic event occurs whenever an event E_2 occurs in the interval formed by events E_1 and E_3. This is formally defined as the non-occurrence of the event E_3 as $\neg O_{in}\,(E_3, [t+1, t_{e1}])$. In order to extend this to hold for recent context this condition ($\nexists$ $E_1'\,[t_{start,}\, t_{end}]\,|\,(t < t_{end} \le t_{e1})$) is added. This condition specifies that, there should not be any occurrence of the event E_1 from the event history E_1 [H] in the interval (t, t_{e1}]. This formal definition is explained using the example shown Fig. 9. When event e_2^2 occurs there are two events in the event history E_1 [H]. Event e_2^2 cannot pair with event e_1^1 since there is an occurrence of the event e_1^2 in the interval formed by these two events. Event e_2^2 is paired with event e_1^2 since there is no other event from E1 [H] has occurred in this interval. Similarly event e_2^3 is paired with the event e_1^2. Event e_3^1 terminates the "A" event initiated by the event e_1^2. Aperiodic operator detects the following events in recent context: {(e_1^2, e_2^2) [8, 9], (e_1^2, e_2^3) [7, 10]}.

Cumulative Aperiodic Operator in Unrestricted Context: In general, Cumulative Aperiodic event is the cumulative version of the aperiodic operator where all the events occurred in the interval formed by event E_1 and E_3 are accumulated. Event histories can be used for the detection of (A* (E_1, E_2, E_3), [t_{s1}, t_{e1}]) in the unrestricted context defined in Section 3.3 using the event histories E_1 [H] = {(3, 5), (4, 6)}, E_2 [H] = {(1, 2), (8, 9), (7, 10), (11, 12)}, E_3 [H] = {(11, 11)} shown Fig. 9. In the unrestricted context, Aperiodic operator generates the following pairs of events {(e_1^1, e_1^2,

e_2^2, e_2^3, e_3^1) [7, 10]}. In this context all the events are accumulated in the interval formed by events e_1^1 and e_3^1.

Cumulative Aperiodic Operator in Recent Context: The “A*” operator in recent context is formally defined as follows:

$$
\begin{array}{l}
O\ (A\ (E_1, E_2, E_3), [t_{sf}, t_{el}]) \triangleq \\
\quad \{\forall E_3 \in E_3\ [H] \\
\qquad \{O\ (E_3, [t_{sa}, t_{ea}]) \wedge (\nexists\ E_3'\ [t_{s,}\ t_e] \mid (t_e < t_{ea}) \wedge E_3' \in E_3\ [H]) \wedge \\
\qquad \{\forall\ E_1 \in E_1\ [H] \wedge \forall\ E_2 \in E_2\ [H] \wedge (O\ (E_2, [t_{sf}, t_{ef}]) \wedge (\nexists E_2'\ [t_s'{}_,\ t_e'] \mid ((t_s' < t_{sf}) \wedge \\
\qquad (t_e' \le t_{e1})) \wedge E_2' \in E_2\ [H]) \wedge (O\ (E_2, [t_{sl}, t_{el}]) \mid (t_{el} < t_{sa})) \wedge \\
\qquad (\nexists E_2''\ [t_s''{}_,\ t_e''] \mid ((t_{el} < t_e'' < t_{sa}) \wedge (t_s'' \ge t_{sf})) \wedge E_2'' \in E_2\ [H]) \wedge \\
\qquad \exists\ t < t_{sf}\ (O\ (E_1{\downarrow}, t) \wedge (\nexists\ E_1'{\downarrow}, t' \mid (t < t' \le t_{el})))))\}\} \\
\quad \vee\ \forall E_3 \in E_3\ [H] \\
\qquad \{O\ (E_3, [t_{sa}, t_{ea}]) \wedge ((\exists E_3'\ [t_{sb}, t_{eb}] \mid (t_{eb} < t_{ea}) \wedge E_3' \in E_3\ [H]) \wedge \\
\qquad (\nexists E_3''\ [t_{s3}'{}_,\ t_{e3}'] \mid (t_{e3}' > t_{eb}) \wedge (t_{e3}' < t_{ea}) \wedge E_3'' \in E_3\ [H])) \wedge \\
\qquad ((O\ (E_2, [t_{sf}, t_{ef}]) \mid (t_{eb} < t_{sf})) \wedge (\nexists E_2'\ [t_s'{}_,\ t_e'] \mid (t_{eb} < t_s' < t_{sf}) \wedge (t_e' \le t_{e1}) \wedge \\
\qquad E_2' \in E_2\ [H]) \wedge (O\ (E_2, [t_{sl}, t_{el}]) \mid (t_{el} < t_{sa})) \wedge \\
\qquad (\nexists E_2''\ [t_s''{}_,\ t_e''] \mid (t_{el} < t_e'' < t_{el}) \wedge (t_s'' \ge t_{sf})\ E_2'' \in E_2\ [H]) \wedge \\
\qquad \exists\ t_{sb} \le t < t_{sf}\ (O\ (E_1{\downarrow}, t) \wedge (\nexists\ E_1'{\downarrow}, t' \mid (t < t' \le t_{e1}))))\} \\
\quad \}.
\end{array}
$$

The above formal definition has two cases, one to handle the case for the first occurrence of the terminator in the history (as it groups all constituent events up to that point) and the second to handle a terminator when there are other previous terminators in the history. The above definition produces the set of event occurrences in the recent context given any two histories. We will explain the formulation of the definition using the same example and the terms used are explained at the end.

When the event e_3^1 occurs, event histories of events E_1, E_2 and E_3 are E_1 [H] = {(3, 5), (4, 6)}, E_2 [H] = {(1, 2), (8, 9), (7, 10)} and E_3 [H] = {(11, 11)}. Event occurrence e_2^1 does not have any effect on the event detection since there is no initiator. Since there are no other events in E_3 [H], satisfying the condition ($\nexists\ E_3'\ [t_{s,}\ t_e] \mid (t_e < t_{ea})$ and falls into the first case of the definition. Now the condition is that accumulating all E_2 events in the interval formed by events E_1 and E_3. But this depends on the initiator, whether e_1^1 or e_1^2 is the initiator. First, event e_1^1 [3, 5] is taken as the initiator. But event e_1^2 [4, 6] has occurred in the interval formed by e_1^1 [3, 5] and e_2^2 [8, 9] and thus fails to satisfy the condition ($\nexists$ ($E_1'{\downarrow}$, t') | ($t < t' \le t_{e1}$)). As the second option event e_1^2 [4, 6] is taken. This satisfies the above condition and thus acts as the initiator for this “A*” event. Thus the events in the interval [6, 11] formed by events e_1^2 and e_3^1 are accumulated and a cumulative aperiodic event is detected with events (e_1^1, e_2^2, e_2^3, e_3^1).

Terms used in the formal definition are: t_{sf} – Start time of First E_2 , t_{ef} – End time of First E_2, t_{sl} – Start time of Last E_2, t_{el} – End time of Last E_2, t_{sa} – Start time of E_3 which is after Last E_2, t_{ea} – End time of E_3 which is after Last E_2.

These terms are specified only in the second case: t_{sb} – Start time of E_3 which is before First E_2, t_{eb} – End time of E_3 which is before First E_2

5 Conclusions and Future Work

In this paper, we have extended the formal definitions of occurrence-based semantics to recent context. These definitions add constraints that are based on the conditions over initiators, detectors, and terminators that should be satisfied for a particular context. The semantics have been implemented using event histories providing procedural semantics. All operators have been formally defined and implemented for all the contexts (recent, continuous, and cumulative) including the unrestricted context, please refer [24][29] for more details. We are in the process of extending the semantics using the disjoint characterization of composite events since this paper assumes all the events as overlapping as shown in Fig. 5. This work needs to be extended to the distributed event occurrence and detection as well.

References

1. Chakravarthy, S. and D. Mishra, *Snoop: An Expressive Event Specification Language for Active Databases.* Data and Knowledge Engineering, 1994. **14**(10): p. 1–26.
2. Chakravarthy, S., et al., *Composite Events for Active Databases: Semantics, Contexts and Detection*, in *Proc. Int'l. Conf. on Very Large Data Bases VLDB*. 1994: Santiago, Chile. p. 606–617.
3. Anwar, E., L. Maugis, and S. Chakravarthy, *A New Perspective on Rule Support for Object-Oriented Databases*, in *1993 ACM SIGMOD Conf. on Management of Data*. 1993: Washington D.C. p. 99–108.
4. Chakravarthy, S., et al., *Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules.* Information and Software Technology, 1994. **36**(9): p. 559–568.
5. Chakravarthy, S., et al. *ECA Rule Integration into an OODBMS: Architecture and Implementation.ICDE*. 1995.
6. Chakravarthy, S., *Early Active Databases: A Capsule Summary.* IEEE Transactions on Knowledge and Data Engineering, 1995. **7**(6): p. 1008–1011.
7. Galton, A. and J. Augusto, *Event detection and Event Definition*. Technical Report 401, Dept. of Comp. Sci., University of Exeter, 2001.
8. Gehani, N., H.V. Jagadish, and O. Shumeli, *Composite Event Specification in Active Databases: Model and Implementation*, in *Proc. 18th Int'l Conf. on Very Large Data Bases.* 1992: Vancouver, Canada.
9. Gehani, N.H., H.V. Jagadish, and O. Shmueli, *Event Specification in an Object-Oriented Database*. 1992: San Diego, CA. p. 81–90.
10. Gatziu, S. and K.R. Dittrich, *Events in an Object-Oriented Database System*, in *Proc. of the 1st Intl Conference on Rules in Database Systems*. 1993.
11. Gatziu, S. and K. Dittrich, *Detecting Composite Events in Active Database Systems Using Petri Nets*, in *IEEE RIDE Proc. 4th Int'l. Workshop on Research Issues in Data Engineering*. 1994: Houston, Texas, USA.
12. Diaz, O., N. Paton, and P. Gray, *Rule Management in Object-Oriented Databases: A Unified Approach*, in *Proceedings 17th International Conference on Very Large Data Bases.* 1991: Barcelona (Catalonia, Spain).
13. Paton, N., et al., *Dimensions of Active Behaviour*, in *Rules in Database Systems.*, N. Paton and M. Williams, Editors. 1993, Springer. p. 40–57.

14. Berndtsson, M. and B. Lings, *On Developing Reactive Object-Oriented Databases.* IEEE Bulletin of the Technical Committee on Data Engineering, 1992. **15**(1-4): pages 31–34.
15. Engstrom, H., M. Berndtsson, and B. Lings, *{ACOOD} Essentials*. 1997, University of Skovde.
16. Bertino, E., E. Ferrari, and G. Guerrini. *An Approach to model and query event-based temporal data*. in *Proceedings of TIME '98*. 1998.
17. Alejandro, P.B., D. Alin, and Z. Juergen, *The REACH Active OODBMS*. 1995, Technical University Darmstadt.
18. Buchman, A.P., et al., *REACH: A Real-Time, Active and Heterogeneous Mediator System.* IEEE Bulletin of the Technical Committee on Data Engineering, 1992.**15**(1–4).
19. Buchman, A.P., et al., *Rules in an Open System: The REACH Rule System*, in *Rules in Database Systems.*,N. Paton and M.Williams,Editors.1993, Springer. p.111–126.
20. Allen, J., *Towards a general Theroy of action and time.* Artificial Intelligence, 1984. **23**(1): p. 23–54.
21. Allen, J. and G. Gerguson, *Action and Evemts in Interval Temporal Logic.* Journal of Logic and Computation, 1994. **4**(5): p. 31–79.
22. Chakravarthy, S. and D. Mishra, *Towards An Expressive Event Specification Language for Active Databases*, in *Proc. of the 5th International Hong Kong Computer Society Database Workshop on Next generation Database Systems*. 1994: Kowloon Shangri-La, Hong Kong.
23. Lee, H., *Support for Temporal Events in Sentinel: Design, Implementation, and Preprocessing*, in *MS Thesis*. 1996, Database Systems R&D Center CISE University of Florida, Gainesville, FL 32611.
24. Adaikkalavan, R., Chakravarthy, S., *Snoop Event Specification: Formalization, Algorithms, and Implementation using Interval-based Semantics*. 2002, Technical Report, The University of Texas at Arlington,
http://itlab.uta.edu/sharma/Projects/Active/publications.htm.
25. Galton, A. and J. Augusto, *Two Approaches to Event Definition*, In Proceedings of 13th International Conference on Database and Expert Systems Applications (DEXA'02), pp. 547–556, September 2002, Aix en Provence, France.
26. Rönn, P., *Two Approaches to Event Detection in Active Database Systems*, M.S. Dissertation, University of Skövde, http://www.ida.his.se/ida/htbin/exjobb/2001/HS-IDA-MD-01-010
27. Claudia L. R. *Toward Duration-based, Constrained and Dynamic Event Types***,** The Second International Workshop on Active, Real-Time, and Temporal Database Systems ARTDB-97
28. Liebig, C., Cilia, M., Buchmann, A.P., *Event Composition in Time-dependent Distributed Systems*. In Proceedings of the International Conference on Cooperative Information Systems, pp. 70–78, Edinburgh, Scotland.
29. Adaikkalavan, R., Snoop Event Specification: Formalization, Algorithms, and Implementation using Interval-based Semantics. 2002, M.S. Thesis, The University of Texas at Arlington.

Reasoning on Workflow Executions

Gianluigi Greco[1], Antonella Guzzo[1], and Domenico Saccà[1,2]

[1] DEIS, University of Calabria, Via Pietro Bucci 41C, 87036 Rende, Italy
[2] ICAR, CNR, Via Pietro Bucci 41C, 87036 Rende, Italy
{ggreco,guzzo}@si.deis.unical.it
sacca@icar.cnr.it

Abstract. This paper presents a new formalism for modelling workflows schemes which combines a control flow graph representation with simple (i.e., stratified), yet powerful DATALOG rules to express complex properties and constraints on executions. Both the graph representation and the DATALOG rules are mapped into a unique program in $\texttt{DATALOG}^{\texttt{ev!}}$, that is a recent extension of DATALOG for handling events. This mapping enables the designer to simulate the actual behavior of the modeled scheme by fixing an initial state and an execution scenario (i.e., a sequence of executions for the same workflow) and querying the state after such executions. As the scenario includes a certain amount of non-determinism, the designer may also verify under which conditions a given (desirable or undesirable) goal can be eventually achieved.

1 Introduction and Overview of the Proposal

Workflow management systems (WFMs) represent today a key technological infrastructure for effectively managing business processes in several application domains including finance and banking, healthcare, telecommunications, manufacturing and production. Many research works deal with the phase of modeling workflow schemes and several formalisms for specifying structural properties have been already proposed to support the designer in devising all admissible execution scenarios.

Most of such formalisms are based on graphical representations in order to give a simple and intuitive description of the workflow structure. In particular, the most common approach is the use of the *control flow graph*, in which the workflow is represented by a labelled directed graph whose nodes represents the task to be performed, and whose arcs describe the precedences among them. Workflow Management Coalition [16] has proposed a standard graphic notation for representing workflow.

A different type of workflow specification has been proposed in [15,14,8] which is based on *state charts* and uses *triggers* to define *Event Condition Action (ECA)* rules for describing transitions among states. Among other workflow graphical formalisms, we mention *Petri Nets* [12] which have a deep formal foundations, and are profitably used for investigating different interesting properties for the process, such as liveness, and boundness. A recent work [13] uses the

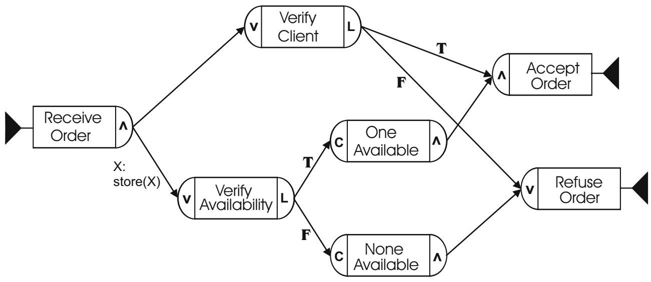


Fig. 1. Example of workflow.

Petri-net theory and tools to analyze workflow graphs. The approach consists in translating workflow graphs into so-called *workflow-nets*, which are are a class of Petri nets tailored towards workflow analysis.

For a better understanding of the main concepts of workflow specification, let us consider the following example, described by a control flow graph, that will be used throughout the rest of the paper. The example describes a typical process for a selling company.

Example 1. A customer issues a request to purchase a certain amount of given product by filling in a request form on the browser (task *ReceiveOrder*). The request is forwarded to the financial department (task *VerifyClient*) and to each company store (task *VerifyAvailability*) in order to verify respectively whether the customer is reliable and whether the requested product is available in the desired amount in one of the stores. The task *ReceiveOrder* will activate both outgoing arcs after completion.

Note that the task *VerifyAvailability*, which is instantiated for each store, either notifies to the task *OneAvailable* that the requested amount is available (label 'T') or otherwise it notifies the non-availability to the task *NoneAvailable* (label 'F'). Observe that the task *OneAvailable* is started as soon as one notification of availability is received whereas the task *NoneAvailable* needs the notifications from all the stores to be activated. The order request will be eventually accepted if both *OneAvailable* has been executed and the task *VerifyClient* has returned the label 'T'; otherwise the order is refused. □

As pointed out by many authors, see e.g [2], the essential limitation of the approach based on the *control flow graph* lies in the ability of specifying *local* dependencies only; indeed, properties such as synchronization, concurrency, or serial execution of tasks, also called in the literature *global constraints*, cannot be expressed. Moreover, in every real world case, there are a number of properties, which cannot be captured by a graph; for instance, in the above example, a natural constraint in every instantiation is that the company will try to satisfy the request by looking at the store nearest to the client, in order to reduce transportation costs. The current trend in workflow management system is to leave unstated all the complex constraints (thus delivering an incomplete specification) or to eventually expressed them using other formalisms, e.g., some form of logics.

In this paper, we propose a logic based environment which combines a graphical representation of workflow schemes with simple (i.e., stratified), yet powerful DATALOG rules to express complex properties and *global constraints* on executions. Both the graph representation and the DATALOG rules are mapped into a unique program in $\mathtt{DATALOG}^{\mathtt{ev!}}$, that is a recent extension of DATALOG for handling events and nondeterministic choices. We must point out that a similar approach has been adopted in [2], by using the *Concurrent Transaction Logic* ($\mathcal{CTR}$) [1], in order to provide a ground to both describe and reason about workflow, by introducing a rich set of constraints. An implementation of the technique is in [9], in which a compiler, named *Apply*, accepts a workflow specification that includes a control graph, the triggers and a set of temporal constraints; as result of the compilation process, an equivalent specification in $\mathcal{CTR}$ is provided.

It is important to note that $\mathcal{CTR}$ logic has been used for solving central problems in the workflow management, such as the **consistency**, i.e., deciding whether a workflow graph is consistent w.r.t. some global constraints, and the **verification**, i.e., deciding whether any legal execution satisfies the global constraints. The framework we propose is, instead, well suited not only for **reasoning** but also for being used as a run-time environment for the **simulation**; in fact, the mapping into $\mathtt{DATALOG}^{\mathtt{ev!}}$enables the designer to simulate the actual behavior of the modelled scheme by fixing an initial state and an execution scenario (i.e., a sequence of executions for the same workflow) and querying the state after such executions. As the scenario includes a certain amount of non-determinism, the designer may also verify under which conditions a given (desirable or undesirable) goal can be eventually achieved.

Example 2. Let us return on the selling company example. A typical scenario of execution consists of a number of requests that are planned in a certain period of time. For instance, the list H of workflow instance activations

$$\mathtt{H} : [\mathtt{init}(\mathtt{id}_1)@(0), \mathtt{init}(\mathtt{id}_2)@(2), \mathtt{init}(\mathtt{id}_3)@(4), \mathtt{init}(\mathtt{id}_4)@(5)]$$

specifies (with the syntax of our language) that 4 orders are planned at times 0, 2, 4, 5. Since for each order, the requested products in the desired quantity are assumed to be taken from a single store, it is obvious that not all the possible schedules will eventually lead to the satisfaction of all orders. An important feature of our language is the powerful querying mechanism, that can be used for planning and scheduling purposes. For instance, by simply entering a query of the form $\langle \mathtt{H}, [\exists\ \mathtt{executed}(\mathtt{id}_4)@(10)], [\mathtt{supplied}(\mathtt{Id}, \mathtt{Store})@(10)]\rangle$, by means of the *goal* $[\exists\ executed(id_4)@(10)]$ we shall ask for a schedule which allows to successfully complete the order id_4 by the time 10 (i.e., we assume that if there is a way to have a successful execution, this will be surely completed within 10 units of time); then, if such a schedule exists, the query will return the names of the stores supplying the items required by each order in the answer relation *supplied*. Obviously, in the case there is no answer, we will be notifies that there is no way to satisfy the order id_4 at the time 10 in the framework of the scenario

H, and, therefore, we can think either of rejecting it in advance, or of planning to stock up with the involved item. □

Moreover, a very important and distinguishing feature of our language is the ability to deal with external events, other then internal ones. For instance, in our running example, natural external events are the closing of a store, and the purchasing of a given quantity of products to assign at a given store. All these events can be naturally modelled by means of very simple and intuitive rules, that can be incrementally inserted into the specifications without modifying any other part.

We stress that contribution of this paper is not only in the proposal of a new formalism: indeed in [5] some of the authors have already shown how to compile a `DATALOG`$^{\mathtt{ev!}}$ program into an equivalent `DATALOG` program with unstratified negation, whose computation can be carried out by means of pre-existent systems for computing models of (disjunctive) logic programs. Presently we are currently on the way of implementing the language into **Disjunctive Datalog System (DLV)** system [7], with the aim of obtaining an effective tool for simulating and reasoning on workflow, based on a widely-used DATALOG programming environment.

2 A Logic Based Environment for Analyzing Workflow

In this section we introduce some basic concepts regarding the specification of a workflow, by focusing the attention both on the *static aspects*, i.e., the description of the relationships among activities not depending from a particular instance using a control flow graph, and on the *dynamic aspects*, i.e., the description of workflow instances whose actual executions depends on status of the system (available servers and other resources).

Definition 1. A *workflow schema* $\mathcal{WS}$ is a directed graph whose nodes are the tasks and the arcs are their precedences. More precisely, $\mathcal{WS}$ is defined as a tuple $\langle A, E, a_0, F, A^{\wedge}_{in}, A^{\vee}_{in}, A^{c}_{in}, A^{\wedge}_{out}, A^{\vee}_{out}, A^{L}_{out}, E^{Q}, E^{L}, \lambda, L\rangle$ where

– A is the set of tasks, $a_0 \in A$ is the initial task, $F \subseteq A$ is the set of final tasks and L is a set of labels; after completion, each task returns a value in L or "fail" in case of an abnormal execution;
– $E \subseteq (A - F) \times (A - \{a_0\})$ is an acyclic relation of precedences among tasks; after its completion, a task *activates* some (or all) of its outgoing arcs, and in turn a task is *started* if some (or all) of its incoming arcs are activated, according to the properties of the involved tasks and arcs, as described next;
– $E^Q \subseteq E$ denotes a set of quantified arcs; each arc $(a, b) \in E^Q$ has associated a formula $\forall X : p(X)$, where $p(X)$ is a relation storing the state of the workflow execution, and the task b has no other incoming arc; once activated, (a, b) instantiates several instances of the task b, one for each value assigned to X, and all such tasks are immediately started;

– $E^L = \{(a,b) | (a,b) \in E \wedge a \in A^L_{out}\}$ are all the arcs leaving the tasks in A^L_{out} and λ is a labelling function from E^L to L; an arc $(a,b) \in E^L$, say with label o, is activated after the completion of the task a only if the output returned by a coincides with o;
– $A^{\wedge}_{in} \subseteq A - \{a_0\}$ are the tasks which act as synchronizer (also called a *and-join* tasks in the literature), thus, a task in $A^{\wedge}_{in}$ cannot be started until after all its incoming arcs are activated;
– $A^{\vee}_{in} \subseteq A - \{a_0\}$ are the tasks which can be started as soon as at least one of its incoming arcs is activated (*or-join*);
– $A^{c}_{in} \subseteq A - \{a_0\}$ are the tasks whose conditions for starting involve more elaborated properties on their incoming arcs (*conditional-join*) – moreover, $A - \{a_0\} = A^{\wedge}_{in} \cup A^{\vee}_{in} \cup A^{c}_{in}$;
– $A^{\wedge}_{out} \subseteq A - F$ are the tasks which activate all their outgoing arcs;
– $A^{\vee}_{out} \subseteq A - F$ are the tasks which activate exactly one of their outgoing arcs, that is non-deterministically chosen;
– $A^{L}_{out} \subseteq A - F$ are the tasks which activate those outgoing arcs whose labels coincide with the label returned by them after completion – moreover, $A - F = A^{\wedge}_{out} \cup A^{\vee}_{out} \cup A^{L}_{out}$. □

Whenever it will be clear by the context, a workflow schema is simply denoted by $\mathcal{WS} = \langle A,\, E,\, a_0,\, F\rangle$.

A workflow schema $\mathcal{WS}$ is represented by suitable tuples in the database $\mathbf{DB}_{CF}(\mathcal{WS})$, called *control flow* database, whose relation schemes are: `task(A)`, `arc(A, A)`, `startTask(a`$_0$`)`, `finalTask(F)`, `qArc(A, A)`, `inOR(`$\mathtt{A}^{\vee}_{\mathtt{in}}$`)`, `inAND(`$\mathtt{A}^{\wedge}_{\mathtt{in}}$`)`, `inCond(`$\mathtt{A}^{\mathtt{c}}_{\mathtt{in}}$`)`, `outOR(`$\mathtt{A}^{\vee}_{\mathtt{out}}$`)`, `outAND(`$\mathtt{A}^{\wedge}_{\mathtt{out}}$`)`, `outLabel(`$\mathtt{A}^{\mathtt{L}}_{\mathtt{out}}$`)`, `lArc(`$\mathtt{A}^{\mathtt{L}}_{\mathtt{out}}$`, A, L)`.

Example 3. The *control flow* database of the workflow schema of Example 1 is organized as follows. The relation `task` contains 7 tuples (one for each node), the tuple in `startTask` is (`ReceiveOrder`) and the tuples in `finalTask` are (`RefuseOrder`) and (`AcceptOrder`). The relation `inAND` identifies the unique and-join node with the tuple (`AcceptOrder`); `inOR` stores the or-join nodes with the tuples (`VerifyClient`), (`VerifyAvailability`) and (`RefuseOrder`); the tuples (`OneAvailable`) and (`NoneAvailable`) in `inCond` identify the conditional-join nodes. The characteristics of the nodes w.r.t. outgoing arcs are stated by the tuple (`ReceiveOrder`) in `outAND` and the tuples (`VerifyClient`) and (`VerifyAvailability`) in `outLabel`. Moreover, we insert the tuples (`OneAvailable`) and (`NoneAvailable`) in the relation `outAND` but they could instead be included in `outOR` as they have only one outgoing arc. Concerning the arcs, they are represented by 8 tuples in the relation *arc*. In addition, the tuple (`ReceiveOrder`, `VerifyAvailability`) in the relation `qArc` states that the corresponding arc is quantified, i.e., when an order is received, then the availability of the required item is verified in every store. Quantification is hence a very succinct way for specifying the same conditions on several activities. Finally, the labelled arcs are identified by the following tuples in `lArc`: (`VerifyAvailability`,`OneAvailable`),(`VerifyAvailability`,`NoneAvailable`), (`VerifyClient`, `AcceptOrder`), (`VerifyClient`, `RefuseOrder`). □

The second aspect of the specification of a workflow concerns the dynamic properties of a workflow, i.e., the the description of how the tasks must be executed.

Definition 2. Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema. Then, we denote by $\mathcal{S}_{ws}$ the set of of *servers* (human and/or computers) who are appointed to execute various tasks. Moreover, the functions $\texttt{task} : \mathcal{S}_{ws} \mapsto A$, $\texttt{duration} : \mathcal{S}_{ws} \mapsto \mathbb{N}$, $\texttt{stateServer} : \mathcal{S}_{ws} \mapsto \{\textit{available}, \textit{busy}, \textit{outOfOrder}\}$ assign to each server the task that it may execute, the time required for the execution, and its current state, respectively. □

The relation `executable(Server, Task, Duration)` stores data about the servers, while the state is registered into the relation `outOfOrder(Server)` whose tuples are added or removed during the executions; the other states are derived using simple rules as we shall show later in this section.

In particular, given an instantiation of the workflow, the state of a server is *busy* if it is executing a certain task, is *outOfOrder* if it is not allowed to execute any tasks, and is *available* if it is waiting for a task to execute. It is worth noting, that also the tasks are characterized by a particular state.

Definition 3. Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema. Then, the function $\texttt{stateTask} : A \mapsto \{\textit{idle}, \textit{ready}, \textit{running}, \textit{ended}\}$ determines the current state of each task, which may take one of the following values:

1. *idle*, thus the task is not yet started as it needs some incoming arc to be activated;
2. *ready*, i.e., the task is ready for execution and has been started but it is waiting for the assignment of a server;
3. *running*, i.e., the task is currently executed by a server;
4. *ended*, i.e., the task has been terminated. □

Given a workflow instance *ID*, the database $\mathbf{DB}_{WE}(\mathit{ID})$ keeps trace of the state of the execution by means of the following relations:

– `startReady(ID, Task, Quantifiers, Time)`, storing the time when the `Task` was started; observe that `Quantifiers` is a stack of values that are used to instantiate tasks activated by quantified arcs – as several quantified arcs (possibly none) may be activated in cascade, their values are collected in a stack (possibly empty);
– `startRunning(ID, Task, Quantifiers, Server, Time)`, storing the time a server has started its execution;
– `ended(ID, Task, Quantifiers, Time, Output)`, storing the `Time` when the execution of `Task` is completed and the `Output` of the execution.

As said before, the state of a task can be derived using simple `DATALOG` rules:

```
state(ID, Task, Quantifiers, ready)   ← startReady(ID, Task, Quantifiers, _),
                                        ¬ startRunning(ID, Task, Quantifiers, _, _).
state(ID, Task, Quantifiers, running) ← ¬ ended(ID, Task, Quantifiers, _, _),
                                     startRunning(ID, Task, Quantifiers, Server_).
```

To simplify the notation, we used some syntactic sugar for writing negative literals in the body of the first of the above rules: $\neg a(X)$, stands for $\neg a'(Y)$, where a' is defined by the new rule: $a'(Y) \leftarrow a(X)$, and Y is the list of all non-anonymous variables occurring in X. We shall use this notation also in the rest of the paper. The following `DATALOG` rule is used to derive whether a server is available to start some task:

$$\begin{aligned}\texttt{available}(\texttt{Server}) \leftarrow\ & \texttt{executable}(\texttt{Server}, \_, \_),\ \neg\, \texttt{outOfOrder}(\texttt{Server}),\\ & \neg\,(\,\texttt{startRunning}(\texttt{ID}, \texttt{Task}, \texttt{Quantifiers}, \texttt{Server}, \_),\\ & \ \neg\, \texttt{ended}(\texttt{ID}, \texttt{Task}, \texttt{Quantifiers}, \_, \_)\,).\end{aligned}$$

In the above rule we have further simplified the notation for writing negated conjunctions in the body of a rule r: $\neg(C)$, where C is a conjunction, stands for $\neg c(X)$, where c is defined by the new rule: $c(X) \leftarrow C$, and X is the list of all variables occurring in C which also occur in r. In the above example C is the conjunction of `startRunning` and $\neg$ `ended`.

In addition to the control flow database $\mathbf{DB}_{CF}(\mathcal{WS})$, each workflow has also associated an internal database $\mathbf{DB}_{I}(\mathcal{WS})$ storing information to be shared among the different instances, and used for computations purposes.

Example 4. In our running example, the company may want to store information about the quantity of products (**QTY**) available in each store into the relations: `store(IDstore, City)`, `product(IDitem)`, and `availability(IDstore, IDitem, QTY)`. Note that the relation `store` is the one used in the quantified arc from `ReceiveOrder` to `VerifyAvailability`. Finally a relation `order(ID, IDitem, QTY)` may be used for storing the quantity of a product required in a workflow instantiation by a customer living in a city. □

2.1 The Overall Architecture

The approach we are proposing is summarized in Figure 2.1; we make use of three databases:

- $\mathbf{DB}_{CF}(\mathcal{WS})$, storing the control flow structure,
- $\mathbf{DB}_{WE}(ID)$, storing information on the instance evolution, such as the status of the tasks and of the servers, and

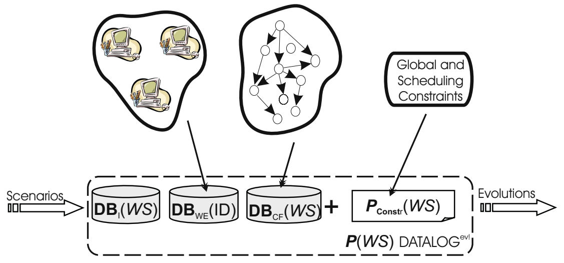


Fig. 2. The run-time environment for workflow executions and simulations.

– $\mathbf{DB}_I(\mathcal{WS})$, storing additional information needed to the execution.

Note that, $\mathbf{DB}_{CF}(\mathcal{WS})$ and $\mathbf{DB}_I(\mathcal{WS})$ are shared among the different instances. Details on each component will be given below.

Moreover, all the *global constraints* and additional constraints on the scheduling of the activities can be translated into a $\mathtt{DATALOG}^{\mathrm{ev!}}$ program $\mathbf{P_{Constr}}(\mathcal{WS})$ over the predicates contained in the above databases. Finally, the run-time execution mechanism can be also defined in term of $\mathtt{DATALOG}^{\mathrm{ev!}}$ rules ($\mathbf{P}(\mathcal{WS})$). As for the usage of the language in this paper, a few intuitions and a number of exemplifications will be sufficient to grasp its semantics (or at least we hope so); nonetheless, the interested reader may find much more details and additional examples in [5].

3 Describing the Workflow Evolution in $\mathtt{DATALOG}^{\mathrm{ev!}}$

The aim of this section is to present a logic framework for the specification of the executions of a workflow for a given scenario of instances. This framework, shown in Figure 3, can be thought of as a simulation environment for workflow executions, and, as the careful reader will notice, it is quite intuitive, declarative in the spirit, yet so powerful to cover all the features of current workflow systems. Moreover, in the following we also show how to extend this framework in order to deal with complex specifications, and how to reason (e.g., scheduling or planning) over them.

Definition 4. Let $\mathcal{WS}$ be a workflow schema, and *ID* be an instance. Then, the logic program $\mathcal{P}(\mathcal{WS})$ modelling the behavior of a workflow system, consists of the facts in $\mathbf{DB}_{CF}(\mathcal{WS})$, and of the set of $\mathtt{DATALOG}^{\mathrm{ev!}}$ rules $\{r_1, \cdots, r_6\}$, shown in Figure 3, over the schema $\mathbf{DB}_{WE}(ID) \cup \mathbf{DB}_I(\mathcal{WS})$. □

Before providing more details on the framework, we point out that bold predicates in Figure 3 represent predicates that can be customized, i.e., specialized for the need of any particular workflow.

The first event, called `init`, is an external event which starts a new workflow instance at a certain time, and triggers the event `run()@(T)`. Each time a run occurs, in the rule r_2, the system tries to assign the ready tasks to the available servers – as we do not use a particular policy for scheduling the servers, the assignment is made in a nondeterministic way. The predicate $\mathtt{unsat}_{ID}()$ is true if it has been already checked that the workflow instance does not satisfy possible constraints on the overall execution – this check is performed during the event `complete`, described below. The predicate $\mathit{executed}_{ID}()$ is true if the workflow instance has already entered a final state so that no other task need to be performed. Once the tasks are assigned to servers, their executions start, and an event `evaluate` is triggered. In the rule r_3, the predicate $\mathbf{evaluation}_{ID}(\mathtt{Task}, \mathtt{L}, \mathtt{Output})$ is used to model the function performed by each task, typically depending on both the execution and internal databases – this predicate must be suitably specified by the workflow designer. The event for

```
r1 : [init(ID)@(T)]
       run()++,
       started_ID(), startReady_ID(ST, [ ], T) ← startTask(ST).
r2 : [run()@(T)]
       evaluate_ID(Task, L, Duration)++,
       startRunning_ID(Task, L, Server, T)    ← ¬unsat_ID(), ¬executed_ID(),
                                                  state_ID(Task, L, Ready), available(Server),
                                                  executable(Server, Task, Duration)
                                                  ⊗ choice((Task), (Server))
                                                  ⊗ choice((Server), (Task)).
r3 : [evaluate_ID(Task, L, D)@(T)]
       complete_ID(Task, L, OG)+(D),          ← evaluation_ID(Task, L, Output).
r4 : [complete_ID(Task, L, Output)@(T)]
       run()++, ended_ID(Task, L, T, Output).
       unsat_ID()                             ← unsatGC_ID(Task, L).
       executed_ID()                          ← finalTask(Task), ¬ unsatGC_ID(Task, L).
       activateArc_ID(Task, L, Next)++        ← outOR(Task), Output ≠ "fail", arc(Task, Next)
                                                  ⊗ ChoiceAny().
       activateArc_ID(Task, L, Next)++        ← outAND(Task), Output ≠ "fail", arc(Task, Next).
       activateArc_ID(Task, L, Next)++        ← outLabel(Task), Output ≠ "fail",
                                                  arcLabel(Task, Next, Label), Label = Output.
r5 : [activateArc_ID(Task, L, Next)@(T)]
       checkNext_ID(Next, [X|L])++
       activatedArc_ID(Task, Next, [X|L])     ← qArc(Task, Next), evaluateQ(Task, Next, X).
       checkNext_ID(Next, L)++
       activatedArc_ID(Task, Next, L)         ← ¬qArc(Task, Next).
r6 : [checkNext_ID(Task, L)@(T)]
       run()++                                ← .
       startReady_ID(Task, L, T)              ← inOr(Task), ¬state_ID(Task, L, ready).
       startReady_ID(Task, L, T)              ← inAnd(Task),
                                                  ¬ (arc(Prec, Task),
                                                  ¬ notifiedArc_ID(Prec, Task, L)).
       startReady_ID(Task, L′, T)             ← inCond(Task), ¬state_ID(Task, L, ready),
                                                  declaredInCond_ID(Task, L, L′).
```

Fig. 3. DATALOG$^{\text{ev!}}$ program $\mathcal{P}(\mathcal{WS})$, modelling the behavior of a workflow system.

completing the task is triggered at the time $\mathtt{T} + \mathtt{D}$, where D is the duration of the task for the assigned server.

Example 5. In our example, the task *VerifyAvailability* must check whether a given store contains enough quantify of the required item. The task behavior is captured by the following rules:

$$\begin{array}{ll}
\textbf{evaluation}_{ID}(\mathtt{VerifyAvailability}, [\mathtt{X}], \text{“T”}) & \leftarrow \mathtt{store(X)},\ \mathtt{order}_{ID}\mathtt{(Item, QTY)}, \\
 & \quad \mathtt{enoughItem(Store, Item, QTY)}. \\
\textbf{evaluation}_{ID}(\mathtt{VerifyAvailability}, [\mathtt{X}], \text{“F”}) & \leftarrow \mathtt{store(X)},\ \mathtt{order}_{ID}\mathtt{(Item, QTY)}, \\
 & \quad \neg\ \mathtt{enoughItem(X, Item, QTY)}. \\
\mathtt{enoughItem(Store, Item, QTY)} & \leftarrow \mathtt{availability(X, Item, QTYavail)}, \\
 & \quad \mathtt{QTYavail} \geq \mathtt{QTY}.
\end{array}$$

Note that these rules make use of the internal database $\mathbf{DB}_I(\mathcal{WS})$. □

After the completion of a task, in the rule r_4, the selection of which of its outgoing arcs be activated depends on whether the task is in $A^{\vee}_{out}$, $A^{\wedge}_{out}$, or A^{L}_{out} and can be done only if the task execution is not failed. The two actions of registering data about the completion and of triggering the event run to possibly assign the server to another task are performed in all cases. The fact $\mathtt{unsat}_{ID}(\mathtt{T})$ is added only if the predicate $\mathbf{unsatGC}_{ID}(\mathtt{Task}, \mathtt{L})$ is true. This predicate is defined by the workflow designer to enforce possible *global constraints* – if not defined then no global constraints are checked after the completion of the task. We shall return on the definition of this predicate for typical global constraints in the next section. Observe that in the case of a final task, if the global constraints are satisfied then we can register the successful execution of the workflow instance. The event `activateArc` performs two distinct actions, depending on whether the arc involved in the notification is quantified or not. In the latter case, the arc is immediately activated. Otherwise, first the quantification is evaluated by means of the user defined predicate $\mathtt{evaluateQ}(\mathtt{Task}, \mathtt{Next}, \mathtt{X})$ and, then, for each value x returned for the variable X, a new instantiation of the arc is both created and activated by pushing x into the stack L. Each activated arc is then stored into the relation $\mathtt{activatedArc}_{ID}(\mathtt{Task}, \mathtt{Next}, \mathtt{L})$. Finally the event `activatedArc` (rule r_5) is triggered in order to the verify whether the activation will make some task ready for starting.

Example 6. In our running example, the quantification for the arc from the task *ReceiveOrder* to *VerifyAvailability* is defined by the following rule:

$$\mathbf{evaluateQ}(\text{“}\mathtt{ReceiveOrder}\text{”}, \text{“}\mathtt{VerifyAvailability}\text{”}, \mathtt{X}) \leftarrow \mathtt{store}(\mathtt{X}).$$

stating that the task *VerifyAvailability* must be “multiplied” for each store `X` of the company. □

The event `checkNext` (rule r_6) stores each task which starts now and triggers the event `run` in order to possibly assign a server for its execution. The predicate $\mathtt{declaredInCond}_{ID}(\mathtt{Task}, \mathtt{L}, \mathtt{L}')$ is to be defined by the workflow designer.

4 Additional Features

In this section, we complete the model by showing how to specify global constraints, and additional internal and external events. First, we formalize the types of constraints we want to model.

Definition 5. Given a workflow $\mathcal{WS}$, a *global constraint* over $\mathcal{WS}$ is defined as follows:

– for any $a \in A$, $!a$ (resp. $\neg !a$) is a *positive* (resp., *negative*) *primitive* global constraint,
– given two positive primitive global constraints c_1 and c_2, $c_1 \prec c_2$ is a *serial* global constraint,
– given any two global constraints c_1 and c_2, $c_1 \vee c_2$ and $c_1 \wedge c_2$ are *complex* global constraints. □

Informally, a *positive* (resp., *negative*) *primitive* global constraint specifies that a task must (resp., must not) be performed in any workflow instance – obviously a negative constraints makes sense only as a sub-expression of a complex global constraint. A *serial* global constraint $c_1 \prec c_2$ specifies that the event specified in the global constraint c_1 must happen before the one specified in c_2. The semantics of the operators $\vee$ and $\wedge$ are the usual. Global constraints can be also mapped into a set of $\texttt{DATALOG}^{\texttt{ev!}}$ rules as follows:

Definition 6. Let $\mathcal{WS}$ be a workflow schema, and $\textbf{Constr}_{ws}$ be a set of global constraint. Then, the program $\mathcal{P}_{\textbf{Constr}}(\mathcal{WS})$ is defined as follows:

– for each global constraint $c = !a$, we introduce the rules:

$$\begin{array}{l}\texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow \texttt{ended}_{ID}(\texttt{a},\_,\_,\texttt{O}), \texttt{O} = \text{``}\texttt{fail}\text{''}.\\ \texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow \neg\, \texttt{ended}_{ID}(\texttt{a},\_,\_,\_).\end{array}$$

where $\texttt{gs}$ equals $\texttt{s}$ if c only occurs as sub-expression of a complex global constraint; otherwise (i.e., c is a global constraint), gs holds $\texttt{g}$.

– for each global constraint $c = \neg\, !a$, we introduce the rule:

$$\texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow \texttt{ended}_{ID}(\texttt{a},\_,\_,\texttt{O}), \texttt{O} \neq \text{``}\texttt{fail}\text{''}.$$

– the rules for a global constraint $c = !a_1 \prec !a_2$ are:

$$\begin{array}{rl}\texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow & \texttt{ended}_{ID}(\texttt{a}_2,\_,\_,\texttt{O2}),\ \texttt{O2} \neq \text{``}\texttt{fail}\text{''}\\ & \texttt{ended}_{ID}(\texttt{a}_1,\_,\_,\text{``}\texttt{fail}\text{''}).\\ \texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow & \texttt{ended}_{ID}(\texttt{a}_2,\_,\texttt{T2},\texttt{O2}), \texttt{O2} \neq \text{``}\texttt{fail}\text{''}\\ & \texttt{ended}_{ID}(\texttt{a}_1,\_,\texttt{T1},\texttt{O1}),\ \texttt{O1} \neq \text{``}\texttt{fail}\text{''},\ \texttt{T2} < \texttt{T1}.\end{array}$$

– for each global constraint $c:\ c_1 \vee c_2$, we have the rule:

$$\texttt{unsatGC1}_{ID}(\texttt{c},\texttt{gs}) \leftarrow \texttt{unsatGC1}_{ID}(\texttt{c}_1,\_),\ \texttt{unsatGC1}_{ID}(\texttt{c}_2.\_).$$

– for each global constraint $c:\ c_1 \wedge c_2$, the rules are:

$$\begin{array}{l}\texttt{unsatGC1}_{ID}(\texttt{c}) \leftarrow \texttt{unsatGC1}_{ID}(\texttt{c}_1,\_)\\ \texttt{unsatGC1}_{ID}(\texttt{c}) \leftarrow \texttt{unsatGC1}_{ID}(\texttt{c}_2,\_).\end{array}$$

□

Let us now define the predicate $\textbf{unsatGC}_{ID}(\texttt{Task},\texttt{L})$ used inside the event `complete`. The problem is selecting the time for checking global constraints. Obviously this check must be done after the completion of a final task. So we can use the following definition :

$$\textbf{unsatGC}_{ID}(\texttt{Task},\texttt{L}) \leftarrow \texttt{finalTask}_{ID}(\texttt{Task}),\ \texttt{unsatGC1}_{ID}(\_,\texttt{g}).$$

Observe that we do not check satisfaction for constraints which are only used as sub-expressions; moreover, we point out that some global constraint check can be anticipated. For instance, the global constraint $c = !a_1 \prec !a_2$ can be checked just after the execution of the task $a2$; so we may introduce the rule:

$$\textbf{unsatGC}_{ID}(\texttt{a}_2,\texttt{L}) \leftarrow \texttt{unsatGC1}_{ID}(\texttt{c},\texttt{g}).$$

An interesting optimization issue is to find out which global constraints could be effectively tested after the completion of each task.

Observe that, as discussed in the previous section, a successful or unsuccessful completion for a workflow instance *ID* is registered by means of the predicate $\mathtt{executed}_{ID}()$ or $\mathtt{unsat}_{ID}()$, respectively. If both predicates are not true, then the are two case: either (i) the execution is not yet finished for some task is currently ready or running, or (ii) non more tasks are scheduled even though a final task was not reached. The latter case indeed corresponds to an unsuccessful completion of the workflow instance and can be modelled as follows:

$$\begin{aligned}
&\mathtt{failed}_{ID}() &&\leftarrow \mathtt{unsat}_{ID}().\\
&\mathtt{failed}_{ID}() &&\leftarrow \mathtt{started}_{ID}(),\ \neg\,\mathtt{executed}_{ID}(), \neg\,\mathtt{working}_{ID}().\\
&\mathtt{working}_{ID}() &&\leftarrow \mathtt{state}_{ID}(\mathtt{T}, \_, \mathtt{ready})).\\
&\mathtt{working}_{ID}() &&\leftarrow \mathtt{state}_{ID}(\mathtt{T}, \_, \mathtt{running})).
\end{aligned}$$

4.1 Side Events

Another aspect we want to consider is the possibility to specify additional internal and external events. The flexibility of $\mathtt{DATALOG}^{\mathrm{ev!}}$ makes these additional specifications quite easy to be modelled; for instance, the rules

$$\begin{aligned}
\mathtt{r}' :\ &[\mathtt{complete}_{ID}(\mathtt{AcceptOrder}, [\mathtt{X}], \mathtt{Output})@(\mathtt{T})]\\
&\mathtt{availability}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYavail} - \mathtt{QTY}),\\
&-\mathtt{availability}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYavail}) \qquad \leftarrow \mathtt{store}(\mathtt{X}),\ \mathtt{order}_{ID}(\mathtt{Item}, \mathtt{QTY}).\\
\mathtt{r}'' :\ &[\mathtt{updateQuantity}_{ID}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYadd})@(\mathtt{T})]\\
&\mathtt{availability}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYavail} + \mathtt{QTYadd}),\\
&-\mathtt{availability}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYavail}) \qquad \leftarrow \mathtt{availability}(\mathtt{X}, \mathtt{Item}, \mathtt{QTYavail}).
\end{aligned}$$

specifies how to update the quantity of items in a store, after the order is accepted, and how to update in general such quantity, in the case a purchase has been done.

Other events may regard the availability of the servers. In the previous section we have seen that each server can be i) *available*, ii) *busy*, or iii) *outOfOrder*. The most simple policy, adopted in the program $\mathcal{P}(\mathcal{WS})$ is that the task being executed by the server is stopped and resumed from the beginning by eventually assigning it to another server. Anyhow, very simple rules can be used for dynamically add or remove a certain number of servers.

5 Querying for an Evolution

Once introduced a model for specifying structural and dynamic aspects of a workflow, the next step is to provide a mechanism for querying the model in order to obtain information on its (possible) evolutions. For instance, in our running, the designer may be interested in knowing whether (and when) a given task has been executed for a given pre-defined scenario.

The scenario is modelled by means of a list containing all the requests (with the corresponding arrival times), and it is denoted with ($\mathcal{S}$). For instance, the scenario $[\mathtt{init(id_1)@(0)}, \mathtt{init(id_2)@(2)}, \mathtt{init(id_3)@(4)}, \mathtt{init(id_4)@(5)}]$ specifies 4 instantiations of the workflow at the times 0, 2, 4 and 5. This scenario is used for querying the $\mathcal{P}(\mathcal{WS})$ $\mathtt{DATALOG^{ev!}}$ program.

Roughly, a query is a triple $\langle \mathtt{S}, \mathtt{G}, \mathtt{R} \rangle$, where $\mathtt{S}$ is a scenario, which lists all the events envisioned to happen, $\mathtt{G}$ is a goal we want to achieve (e.g., the scheduling of all the orders), and $\mathtt{R}$ is a list of predicate symbols whose tuples will be the result of the query.

Definition 7. A query on a $\mathtt{DATALOG^{ev!}}$ program P is of the form $\langle \mathtt{S}, \mathtt{G}, \mathtt{R} \rangle$ where

- S is a scenario – say that t^S_{max} is the last time of the events in S;
- G (*goal*) is a list of query conditions of the form $[\mathtt{g_1@(t_1)}, \ldots, \mathtt{g_n@(t_n)}]$ ($n \geq 0$), where $0 \leq t_1 < t_2 < \ldots < t_n \leq t^S_{max}$; each g_i can be:
  - $\exists(A)$, where A is a conjunction of ground DB literals;
  - $\forall(A)$, where A is a (possibly negated) conjunction of ground DB literals;
  - $opt(X : A)$, where $opt = min$ or max, X is a variable and A is a conjunction of literals containing no variables except X;
- R (*result*) is a list $[\mathtt{r_1(X_1)@(t_1)}, \ldots, \mathtt{r_m(X_m)@(t_m)}]$, $m > 0$ and $t_1 \leq \ldots \leq t_m \leq t^S_{max}$, where r_i is any predicate symbol, say with arity k_i, and X_m is a list of k_i terms. $\square$

The interested reader may find a more formal definition of the semantics of a query in [5].

Definition 8. Given a query $\mathtt{Q} = \langle \mathtt{S}, \mathtt{G}, \mathtt{R} \rangle$ where $\mathtt{R} = [\mathtt{r_1@(t_1)}, \ldots, \mathtt{r_m@(t_m)}]$, on a $\mathtt{DATALOG^{ev!}}$ program P, and a database D, an (nondeterministic) *answer* of Q on D, denoted by $Q(D)$, is either (i) the list of relations $[\mathbf{r}(X_1)_1, \ldots, \mathbf{r}(X_i)_m]$ such that $\mathbf{r}_i = \{x_i | r'_i(x_i, t_i, max_{subT}\ (M, t_i)) \in M$ and x_i unifies with $X_i\}$, where r' is the temporal version of the predicate symbol r and M is a stable model of the program P or (ii) the empty list if there is no stable model. $\square$

Assume, in our running example, the company has planned to have a number of requests, constituting a scenario H. The designer want to know the possible evolutions; this aim can be achieved by supplying the query $\langle \mathtt{H}, \emptyset, \mathtt{R} \rangle$. Indeed, as no goal is issued, no restriction on executions will be enforced and any of them will be eventually returned.

For the following, let t_{max} be the sum of all the durations of the tasks, declared by any server. Observe that such t_{max} is an upper bound on the completion time of any instance, t^S_{max}.

Example 7. In our running example, assume there are two stores of the company, namely S_1 and S_2. Moreover, S_1 contains 100 units of item i_1 and 50 units of item i_2, while store S_2 contains only 30 units of item i_1, i.e. we have the tuple $\mathtt{availability(S_1, i_1, 100)}$, $\mathtt{availability(S_1, i_2, 50)}$ and $\mathtt{availability}$ $\mathtt{(S_2, i_1, 30)}$. Assume two request are given, id_1 of 40 units of i_1, and id_2 of 30 units of i_1 and 60 of i_2. Then, the query $\langle \mathtt{H}, \mathtt{G}, \mathtt{R} \rangle$, where $\mathtt{H} = [\mathtt{init(id_1)@(0)},$

$\mathtt{init(id_2)@(0)}$], $\mathtt{G}$ = [$\exists$ $\mathtt{executed(id_1)@(t_{max})}$, $\exists$ $\mathtt{executed(id_2)@(t_{max})}$], and $\mathtt{R}$ = [$\mathtt{availability(X, I, Q)@(t_{max})}$], will return an empty result, stating that it is not possible to satisfy both requests.

Conversely, if we assume that an $\mathtt{updateQuantity}_{ID}(\mathtt{X}, \mathtt{i_2}, \mathtt{50})$ event may occur, by inserting such event in $\mathtt{H}$, i.e., if we may purchase 50 items of i_2 to store in a certain store, we obtain a result $\mathtt{availability(S_1, i_1, 30)}$, $\mathtt{availability}$ $\mathtt{(S_1, i_2, 40)}$, $\mathtt{availability(S_2, i_1, 30)}$, $\mathtt{availability(S_2, i_2, 0)}$, corresponding to store the additional quantity of i_2 in the first store. □

Finally, the following proposition, whose proof is omitted due to space limitations, states that our model is sound and complete w.r.t. the satisfaction of the global constraints (the predicate $\mathtt{executed(ID)}$ is the one defined in the previous section).

Proposition 1. *Let* $\mathcal{WS}$ *be a workflow schema,* $\mathbf{Constr}(\mathcal{WS})$ *a set of global constraint, and* Q *be the query* $\langle \mathtt{init(id_1)@(0)}, \exists(\mathtt{executed(id_1)@(t_{max})}, \mathtt{R})\rangle$ *on the program* $\mathcal{P}(\mathcal{WS}) \cup \mathcal{P}_{\mathbf{Constr}}(\mathcal{WS})$. *Then,* R *is empty if and only if there exists no instance that satisfies the given constraints.* □

6 Conclusions and Further Work

We have presented a new formalism which combines a graph representation of workflow schemes with simple (i.e., stratified), yet powerful DATALOG rules to express complex properties and constraints on executions. We have shown that our model can be used as a run-time environment for workflow execution, and as a tool for reasoning on actual scenarios. The latter aspects gives also the designer the ability of finding bugs in the specifications, and of testing the system's behavior in real cases.

On this way, our next goal is to devise workflow systems that automatically fix "improperly working" workflows (typically, a workflow systems supply, at most, warning message when detecty such cases). In order to achieve this aim, we shall investigate formal methods that are able to understand when a workflow system is about to collapse, to identify optimal scheduling of tasks, and to generate improved workflow (starting with a given specification), on the basis of some optimality criterion.

References

1. A. Bonner. Workflow, Transactions, and Datalog. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 294–305, 1999.
2. H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic Based Modeling and Analysis of Workflows. In*Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 25–33, 1998.
3. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2), pages 119–153, 1995.

4. S. Greco, D. Saccà and C. Zaniolo. Extending Stratified Datalog to Capture Complexity Classes Ranging from P to QH. In *Acta Informatica*, 37(10), pages 699–725, 2001.
5. A. Guzzo and D. Saccà. Modelling the Future with Event Choice DATALOG. Proc. AGP Conference,, pages 53–70, September 2002.
6. G. Kappel, P. Lang, S. Rausch-Schott and W. Retschitzagger. Workflow Management Based on Object, Rules, and Roles. *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society*, 18(1), pages 11–18, 1995.
7. Eiter T., Leone N., Mateis C., Pfeifer G. and Scarcello F.. A Deductive System for Non-monotonic Reasoning. *Proc. LPNMR Conf.*, 363–374, 1997.
8. P. Muth, J. Weißenfels, M. Gillmann, and G. Weikum. Integrating Light-Weight Workflow Management Systems within Existing Business Environments. In *Proc. 15th Int. Conf. on Data Engineering*, pages 286–293, 1999.
9. P. Senkul, M. Kifer and I.H. Toroslu. A logical Framework for Scheduling Workflows Under Resource Allocation Constraints. In *VLDB*, pages 694–705, 2002.
10. D. Saccà and C. Zaniolo. Stable Models and Non-Determinism in Logic Programs with Negation. In *Proc. ACM Symp. on Principles of Database Systems*, pages 205–218, 1990.
11. M. P. Singh. Semantical considerations on workflows:An algebra for intertask dependencies. In *Proc. of the Int. Workshop on Database Programming Languages*, pages 6–8, 1995.
12. W. M. P. van der Aalst. The Application of Petri Nets to Worflow Management. *Journal of Circuits, Systems, and Computers*, 8(1), pages 21–66, 1998.
13. W. M. P. van der Aalst, A. Hirnschall and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In *Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering*, pages 534–552, 2002.
14. D. Wodtke, and G. Weikum. A Formal Foundation for Distributed Workflow Execution Based on State Charts. In *Proc. 6th Int. Conf. on Database Theory (ICDT97)*, pages 230–246, 1997.
15. D. Wodtke, J. Weissenfels, G. Weikum, and A. Dittrich. The Mentor project: Steps towards enterprise-wide workflow management. In *Proc. of the IEEE International Conference on Data Engineering*, pages 556–565, 1996.
16. The Workflow Management Coalition, *http://www.wfmc.org/*.
17. Zaniolo, C., Transaction-Conscious Stable Model Semantics for Active Database Rules. In *Proc. Int. Conf. on Deductive Object-Oriented Databases*, 1995.
18. Zaniolo, C., Active Database Rules with Transaction-Conscious Stable Model Semantics. In *Proc. of the Conf. on Deductive Object-Oriented Databases*, pp.55–72, LNCS 1013, Singapore, December 1995.
19. Zaniolo, C., Arni, N., and Ong, K., Negation and Aggregates in Recursive Rules: the LDL++ Approach, *Proc. 3rd Int. Conf. on Deductive and Object-Oriented Databases*, 1993.

Optimization of Storage Structures of Complex Types in Object-Relational Database Systems

Steffen Skatulla and Stefan Dorendorf

Friedrich-Schiller-University Jena, Germany
{skatulla,dorendorf}@informatik.uni-jena.de
www.informatik.uni-jena.de/dbis

Abstract. Modern relational DBMS use more and more object-relational features to store complex objects with nested structures and collection-valued attributes. Thus evolving towards object-relational database management systems. This paper presents results of the project "Object-Relational Database Features and Extensions: Model and Physical Aspects"[1] of the Jena Database Group. It introduces an approach to optimize the physical representation of complex types with respect to the actual workload, mainly based on two concepts: First, different variants of physical representation of complex objects can be described and controlled by a new Physical Representation Definition Language (PRDL). Second a method based on workload capturing is suggested that allows to detect the need for physical restructuring, to evaluate alternative storage structures with respect to better performance and lower execution costs and to get well-founded improvement estimations.

Keywords: object-relational database management system, complex type, storage structure, workload capturing, optimization, clustering

1 Introduction

Today's relational database systems are moving towards support for non-standard applications by offering extensibility and object-oriented features [1,2]. These non-standard applications in fields like engineering and spatial databases need to store instances of complex nested data types with specialized processing logic [3,4,5].

Consequently, SQL has been extended by concepts for the definition of new user-defined data types with structure, inheritance and simple collections [6]. DBMS products are being extended with the capability to store user-defined data types as well [7,8,9]. But only a limited set of modeling concepts is supported as of now: nested structures, single inheritance and methods as well as simple collections like arrays. Advanced collections like sets, multi-sets and lists with arbitrary nesting are not yet supported by the standard and products [10].

[1] The research project is supported by an IBM Shared University Research Grant.

Therefore our group suggested an SQL extension called SQL:1999$^+$ integrating complete collection support into standard SQL [10,11]. These constructs are what we refer to when we talk about complex object structures in this paper. Additionally, in current object-relational database management systems (ORDBMS) there is usually only one fixed way to represent complex objects physically. The user is not allowed to choose from various physical representations with respect to varying access and processing profiles. Nevertheless, the literature suggests a number of reasonable physical representation alternatives to choose from [12,13,14,15].

The objective of this paper is to present methods for the optimization of the physical storage of complex types. These methods are developed to be usable for both, the (DBMS specific) physical database design and the physical reorganization necessary to adapt databases to changing workloads.

During the physical database design storage structures are chosen that promise best performance for a given operational workload which represents the mix of DBMS operations invoked by all applications running on the database during normal production mode. For this purpose we suggest a method to capture access profiles. In order to optimize the physical design the storage structures need to be described explicitly. Therefore we present an appropriate specification language called PRDL [11,16,17]. In productive environments it is impossible to optimize storage structures in try-and-error procedures by changing the physical design until the performance is good enough. We present an analytical approach that allows to calculate execution costs of the captured workload mix with respect to different physical storage structures. Workload capturing, PRDL and analytical cost estimation are key steps on the way to our long-term objective of automatic storage optimization.

Section 2 discusses alternative physical storage structures for complex objects and presents our Physical Representation Definition Language. An example to illustrate physical design alternatives is introduced. Workload capturing and cost-based analytical evaluation methods are discussed in Section 3 and 4. The analysis of alternative storage structures in our example scenario shows the optimization potential and the improvements achievable with our methods. Section 5 summarizes our results and closes with an outlook to further work.

2 Storage of Complex Types in ORDBMS

2.1 Approaches to Complex Object Integration

Object-relational DBMS are trying to extend existing relational database systems with flexible type systems and object support. But traditional normalized data in flat tables has to coexist with complex data objects. A key demand is that the performance of processing traditional simple data is not affected by the object extensions at all.

Consequently the new features have to be integrated into the DBMS in a way that changes the existing implementation as little as possible. One commonly used way is to limit the extensions to the upper tier of the DBMS architecture

[7,18]. Complex type support can be implemented mainly in the query compiler without affecting other components like access, record and buffer management, transaction control, recovery manager and optimizer too much [7].

This way of implementation rises the question how to store objects physically, that is how to map complex data structures to internal flat data records.

2.2 Classifying Alternatives of Physical Representation

The literature suggests a number of variants to store complex objects physically [12,13,14,15]. A simple approach is to serialize objects and to store them in one physical record. This is what products like Oracle (with respect to `VARRAY`s), DB2 and Informix do. Other suggestions are to break up compound objects, to store their sub-objects in different physical records (e.g. Oracle `NESTED TABLE`s) and to cluster resulting object parts in order to support a given access profile. But generally speaking, DBMS products offer so far too little flexibility with respect to physical representation.

In order to approach the issue systematically, we developed a classification scheme of possible storage alternatives for complex objects. We found a number of orthogonal criteria spanning up the space of options. Each criterion is presented here along with a short discussion.

- *Compact object storage vs. object decomposition*
  The question is whether to store a compound object in one physical record allowing fast access to the entire object or to decompose it spreading its sub-objects among different physical records supporting partial object retrieving and sub-object referencing. In the case of object decomposition the object is split up into hierarchically interconnected records. The root record is referred to as primary record, the other records are referred to as secondary records.
- *Storage location of physical records*
  Different types of physical records can be assigned to different storage locations. Thereby we can speed up frequent record accesses by assigning fast devices and reduce costs by assigning slow devices to rarely accessed records.
- *Clustering strategy*
  Data access can be speed up by clustering objects, that is, to store objects frequently retrieved together in close physical locations [15,19]. We differentiate between object-based and object-spanning clustering strategies. Object-based clustering means to store physical records belonging to one object close together. Object-spanning clustering stores physical records of the same type (representing the same sub-structure, e.g. some sub-attributes) of all objects close together spreading objects over different clusters.
- *Efficient long record support in the storage system*
  To support larger collections, the storage system could provide long physical records. Otherwise, the access system on top of it has to split the collection to store the parts in different physical records linked together in some way.
- *Type of references*
  The question is whether to use physical addresses to reference objects and to interconnect the physical records of compound objects yielding fast direct

access at the expense of slower updates or to use value-based references that allow cheaper management but require index look-ups for dereferencing. Alternatively, hybrid references (logical references with physical location hints) could be used, combining advantages of both types at the expense of storage space.

– *Physical collection structure*
  Collections (arrays, lists, sets, multi-sets) defined on the logical level can be represented physically with different structures. For instance a logical set could be represented as a linked list of records or as a tree structure etc. The choice of a physical structure should respect the access characteristics.

2.3 PRDL – Physical Representation Definition Language

Based on the storage classification criteria presented in section 2.2 we developed PRDL [11,16,17]. This language is intended to describe how a given logical object structure is stored physically.

A possible way to define this language is to extend the SQL-DDL. This approach is chosen by major DBMS vendors for the specification of physical table placement and other aspects. However this mixes physical aspects into the SQL-DDL which was originally designed and standardized as logical data definition language. And it leads to vendor specific and non-standard SQL-extensions confusing physical and logical aspects.

Another way is to define a separate language that allows to specify physical storage structures for logical objects. Of course the physical specifications have to refer to the objects defined by the logical SQL-DDL, but the language itself and its grammar can be separated.

For simplicity we decided to append PRDL-specifications to SQL-DDL statements like `CREATE TYPE` and `CREATE TABLE` but to preserve a strict language separation, even though a fully separated definition with adequate references would be possible. Figures 2 and 3 show how PRDL-specifications are appended to a `CREATE TABLE` statement. The specification of physical storage structures follows between `BEGIN PRDL` and `END PRDL`. The language comprises among others the following clauses:

– A clause to define object decompositions into separate physical data sets, mainly to support different access frequencies of object parts. This is done by placing object attributes into secondary records:
  ```
  SECONDARY RECORD (REAL_NUMBER.ELEMENT)
  ```
– A clause to specify physical storage locations:
  ```
  STORE IN TABLESPACE_A ...
  ```
– A clause that allows to rearrange objects and their parts into clusters, to reduce access costs by storing commonly accessed data physically close together in a minimum of data pages:
  ```
  STORE ... IN CLUSTER CL1 (...)
  ```

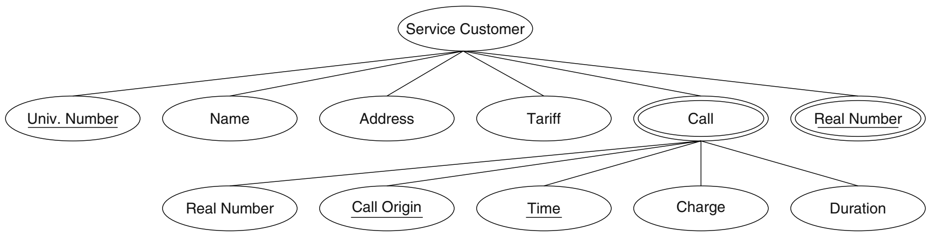


Fig. 1. Example “Telecommunications Service”

To simplify the usage of PRDL, adequate default mappings are defined emulating the physical object mapping strategies of todays ORDBMS products to ensure compatibility.

A detailed description of PRDL would go beyond the scope of this paper. Therefore we refer to [17] and limit our presentation to examples to give an impression.

The main advantages of PRDL are to improve data independence by separating the logical from the physical design and to provide a proper interface language to the ORDBMS component that maps objects to physical storage structures (usually located in the data management layer).

2.4 Example “Telecommunications Service”

The example presented in this section illustrates the specification and usage of alternative physical storage structures in a telecommunications service scenario. We have chosen to focus the example to the specific area of clustering strategies. Clustering promises to reduce I/O costs, thereby increasing performance. This example was chosen for two reasons: On the one hand it demonstrates achievable performance improvements quite well. On the other hand the specific storage structures used can be implemented already with Oracle9i [20] despite the generally not yet complete set of object-relational constructs provided by today’s ORDBMS products.

The example models a nationwide uniform ’virtual’ telephone number as shown in Figure 1. Besides the ’virtual’ number itself and information about its name, address and tariff it comprises two sets. One set of real telephone numbers the calls are routed to and another data set recording all incoming calls to the ’virtual’ number.

Object-Spanning Clustering. Object-spanning clustering aims to store physical records of the same type close together in one location. With respect to the example this means to split up the service customer objects into a number of physical records as drawn in Figure 2 and to store all `SERVICE_CUSTOMER` records close together, all `REAL_NUMBER` records close together and all `CALLS` records close together likewise (indicated by the grey boxes). The corresponding PRDL specification is shown in Figure 2 too.

```
CREATE TABLE SERVICE_CUSTOMER OF SERVICE_CUSTOMER_TYPE
  ...
BEGIN PRDL
  STORE IN TABLESPACE_A
  SECONDARY RECORD (REAL_NUMBER.ELEMENT) IN TABLESPACE_B REFERENCE IS LOGICAL
  SECONDARY RECORD (CALL.ELEMENT) IN TABLESPACE_C REFERENCE IS LOGICAL
END PRDL
```

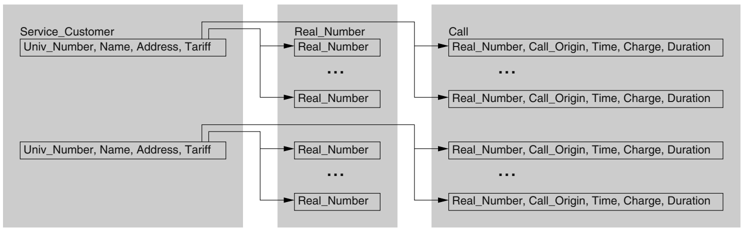

Service_Customer
Univ_Number, Name, Address, Tariff
Real_Number
Real_Number
...
Real_Number
Call
Real_Number, Call_Origin, Time, Charge, Duration
...
Real_Number, Call_Origin, Time, Charge, Duration
Univ_Number, Name, Address, Tariff
Real_Number
...
Real_Number
Real_Number, Call_Origin, Time, Charge, Duration
...
Real_Number, Call_Origin, Time, Charge, Duration

Fig. 2. Object-spanning clustering: PRDL specification and physical structure

```
CREATE TABLE SERVICE_CUSTOMER OF SERVICE_CUSTOMER_TYPE
  ...
BEGIN PRDL
  STORE IN TABLESPACE_A IN CLUSTER CL1 (UNIV_NUMBER)
  SECONDARY RECORD (REAL_NUMBER.ELEMENT) IN PRIMARY RECORD CLUSTER REFERENCE IS HYBRID
  SECONDARY RECORD (CALL.ELEMENT) IN PRIMARY RECORD CLUSTER REFERENCE IS HYBRID
END PRDL
```

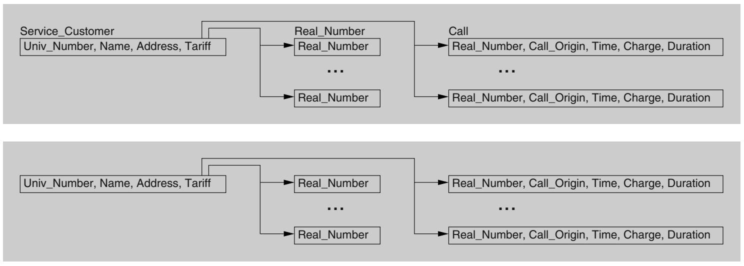

Service_Customer
Univ_Number, Name, Address, Tariff
Real_Number
Real_Number
...
Real_Number
Call
Real_Number, Call_Origin, Time, Charge, Duration
...
Real_Number, Call_Origin, Time, Charge, Duration
Univ_Number, Name, Address, Tariff
Real_Number
...
Real_Number
Real_Number, Call_Origin, Time, Charge, Duration
...
Real_Number, Call_Origin, Time, Charge, Duration

Fig. 3. Object-based clustering: PRDL specification and physical structure

Object-spanning clustering is advantageous mainly in cases where records of the same type are sequentially scanned quite frequently, e.g. scans on real telephone numbers. This is because clustering promises to reduce I/O costs of sequential scans since more matching and less irrelevant data sets are fetched at once per I/O operation. On the other hand this storage strategy is disadvantageous for frequent accesses to objects as a whole because this requires to retrieve both, the primary and secondary records and to rejoin them.

Object-Based Clustering. In contrast to object-spanning clustering this strategy tries to store records of one object physically close together (preferably on one data page) as shown in Figure 3 [21]. On the one hand this offers the possibility to fetch the clustered object parts with one single page access. On the

```
SERVICE_CUSTOMER(UNIV_NUMBER, NAME,...)
REAL_NUMBER(UNIV_NUMBER, REAL_NUMBER) UNIV_NUMBER references SERVICE_CUSTOMER
CALLS(UNIV_NUMBER, REAL_NUMBER, CALL_ORIGIN, TIME,...) UNIV_NUMBER references SERVICE_CUSTOMER
```

Fig. 4. Example tables created in Oracle9i

other hand it becomes possible to create common cluster indexes (1:N-indexes) pointing to the common location of the primary and secondary records of each object [22]. The associated PRDL specification is displayed in Figure 3 too.

This storage structure supports operations best that access entire objects using the common index, because the number of I/O operations is reduced and rejoining is inexpensive. But if just parts of objects are accessed, this variant suffers from a considerable overhead of fetching unneeded data. That applies especially to cases where larger ranges of only one sub-record type need to be scanned (e.g. scanning for all real telephone numbers).

Implementation in Oracle9i. We have implemented the two storage alternatives using Oracle9i in order to demonstrate the performance and cost improvements that can be achieved by optimizing physical storage structures. The example environment was modeled with three interconnected tables as shown in Figure 4. The object-based clustering strategy was implemented using the table clustering feature of Oracle9i [20] that allows to store tuples of several tables with a common cluster key physically close together as shown in Figure 9. In contrast, the object-spanning clustering strategy was implemented without using this feature by placing the tables into different tablespaces.

3 Workload Capturing

To estimate the improvement of complex database object conversion, it is necessary to determine the exact workload. Workload, in our case, is a list of DML statements each associated with its number of executions. In the following sections we will discuss the possibilities and procedures for workload capturing.

3.1 Basic Alternatives for Workload Capturing

One first simple and suitable possibility to evaluate the workload is to use the knowledge of developers and users about the workload. But developers usually only know the behavior of applications in their implementation they were involved. And the information gathered interviewing application users is usually not sufficient because they only know the frequency of tasks done individually. The logic standing behind the application program they often don't know. So this way is in most cases not practicable to create a total access profile of all applications working with the database. Useful ways for creating qualified access profiles for workload capturing must use information from the DBMS as the

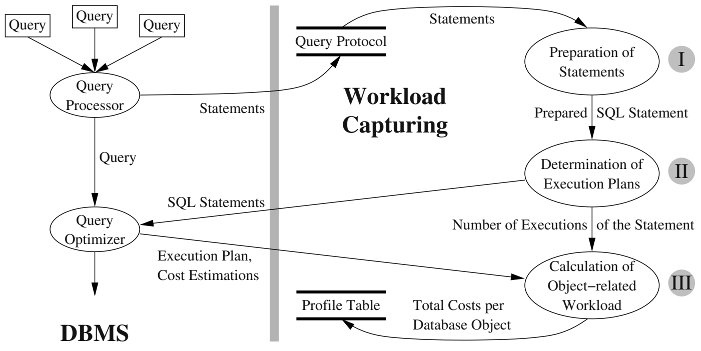

Query
Query
Query
Query Processor
Statements
Query
Query Optimizer
SQL Statements
Execution Plan, Cost Estimations
DBMS
Query Protocol
Statements
Workload Capturing
Preparation of Statements
I
Prepared SQL Statement
Determination of Execution Plans
II
Number of Executions of the Statement
Calculation of Object–related Workload
III
Profile Table
Total Costs per Database Object

Fig. 5. Workload capturing

central instance for the processing and administration of data which has to be made accessible.

One source of information is the transaction log. For our purpose, however, it is not sufficient because retrieval operations that have a large quota on the workload are not recorded there.

Promising sources of information are the components that are part of the instruction processing. As all SQL statements have to pass the query processor first, it is possible to log the statements and the number of their executions. If the DBMS product does not provide an interface to log processed statements a logging component has to be placed between the DBMS and the applications.

A promising source of information is especially the query optimizer. Its task is to find plans to execute DML statements economically by choosing and (re)arranging basic operations [23]. In addition to the execution plan, the query optimizer estimates execution costs. Supposing the model for the cost estimation is sufficiently exact, the information provided by the query optimizer can be used for workload capturing.

Furthermore the logging of accesses on the level of records and access paths would be well suitable. But in contrast to statement logging – more easily realizable on upper layers – record level logging needs modifications on deeper DBMS software layers. The problem is that the interfaces to these internal layers are usually not externally opened and documented by DBMS vendors.

3.2 Workload Capturing by Measurement

Workload capturing by measurement bases on the logging of DML statements. Additionally the number of executions of all statements is recorded in a representative period of time to quantify the workload. To reduce the size of the instruction log each statement is recorded only once, repetitions just increase a counter. For long recording periods log compression methods as described in [24] could be used. The suggested processing steps are shown in Figure 5. If it is necessary to reduce the additional load on the production system these tasks could run on another machine.

```
ST_ID SQL_TEXT                                                           EXECUTIONS
----- ------------------------------------------------------------------ ----------
   54 SELECT * FROM Service_Customers ORDER BY Univ_Number                        1
   44 SELECT Service_Customers.Tariff,Real_Numbers.Real_Number                25000
   44 FROM Service_Customers, Real_Numbers                                    25000
   44 WHERE Service_Customers.Univ_Number=10093 AND                           25000
   44 Service_Customers.Univ_Number=Real_Numbers.Univ_Number                  25000
```

Fig. 6. Logged instructions stored in relational form

```
explain plan for
SELECT Service_Customers.geb_satz, Ral_Numbers.Real_Number
FROM Service_Customers, Real_Numbers WHERE Service_Customers.Univ_Number=10093
AND Service_Customers.Univ_Number=Real_Numbers.Univ_Number

SELECT id, operation, options, object_name, cost FROM plan_table

ID OPERATION            OPTIONS             OBJECT_NAME                COST
-- -------------------- ------------------- --------------------   ----------
 0 SELECT STATEMENT                                                       3
 1 NESTED LOOPS                                                           3
 2 TABLE ACCESS         CLUSTER             SERVICE_CUSTOMERS             2
 3 INDEX                UNIQUE SCAN         I_UNIV_NUMBER                 1
 4 TABLE ACCESS         CLUSTER             REAL_NUMBERS                  1
```

Fig. 7. Production of an execution plan and an excerpt from the plan

```
ST_ID OBJECT_NAME              OPERATION                     COSTS      COUNT
----- ------------------------ ------------------------- ---------- ----------
   54 SERVICE_CUSTOMERS        TABLE ACCESS                     398          1
   44 SERVICE_CUSTOMERS        TABLE ACCESS                       2      25000
   44 I_UNIV_NUMBER            INDEX                              1      25000
```

Fig. 8. A portion of a captured workload

In step I the database administrator has to choose statements relevant for the workload capturing. This selection can at least be partly carried out during the logging, if the used tool allows to confine the logged statements like the Microsoft SQL Server Profiler does. Afterwards the recorded statements have to be prepared, that the query optimizer can process them. Figure 6 shows an excerpt of a relationally stored log produced with our prototype tool for workload capturing implemented for Oracle9i. The statement with the number 44 shows the search for service customer data needed to process a telephone call in our example environment that was processed 25000 times in the observation period.

In step II the prepared SQL statements will be transferred to the query optimizer to find an execution plan with low processing costs. Figure 7 shows the operator listing of an execution plan for the statement shown above. In the example, object-based clustering was used to store service customers close to their real telephone numbers.

The execution plan also includes cost estimations of the query optimizer. Here we have to emphasize that the provided values are estimations and that the exactness of our calculations bases on the exactness of the cost model of the optimizer. To examine (e.g. by measurement) if cost models of optimizers of usual

DBMS products are exact enough is a current task. During this work we also try to get information on the buffering of data that is detailed enough for further integration into the cost models. Another problem we are confronted with at this point is that many features PRDL provides are not (yet) implemented into the available DBMS products. So we often have to use our own cost estimations, or we have to use available alternative storage structures to simulate the features of PRDL. In the future, however, we hope that DBMS products will provide new storage features like those of PRDL.

The next step is to weight the cost estimations of the optimizer with the number of executions of the associated statements as shown in Figure 8 and to sum up the total cost of the logged workload in step III.

3.3 Evaluation of Alternative Storage Structures

Before expensive physical restructuring is done it would be useful to compare the planned new storage structure to the old existing one and to estimate whether a reduction of the total processing costs of the workload can be reached.

To calculate the costs induced by the workload on the new structured data, we just simulate the new storage structures. This is possible because query optimizers normally do not need existing structures but use statistics and other meta information describing the database objects. Only the creation of this meta information with assigned statistics in the database catalog is required for our purpose. There is no need to change real structures and to transfer data. A simulation makes it possible to evaluate different promising physical representations before really changing them. After the new meta information and the associated statistics are created, steps II and III of the workload capturing have to be done to estimate the total execution costs of the recorded workload using the new physical structures. By comparing the total processing costs caused by the actual structures and the alternative structures we are able to estimate the advantage or disadvantage of a change in the physical storage structure of the database objects.

Unfortunately it is quite difficult to create the new meta information and statistics, exact knowledge of the catalog structure and semantics of the used DBMS is necessary. This problem is complicated by the missing of standardization of catalog entries describing physical aspects of the database.

There are different ways to create the needed catalog entries. A possible way is to extend DML statements by a clause (e.g. `WITH STATISTICS ONLY`) allowing to produce catalog entries without really creating tables and indexes as described in [25]. But for our tests we calculated and inserted the necessary statistics manually using well known calculation schemes. In spite of the mentioned problems the creation of the simulation data is less risky than a disadvantageous conversion of the storage structures.

4 Example Environment Examinations

In this section we discuss the potential of storage conversion of database objects utilizing our telecommunications service example. We show how to estimate the effects of storage conversion and demonstrate the feasibility of exact estimations. The purpose is to find out whether it is better to use object-based or object-spanning clustering for the data of our telecommunications example assuming a near realistic workload described below.

First we created service customer data sets with associated real telephone numbers. Then the simulation creates records like those created by the processing of real telephone calls. Thereby the same database access operations will be processed as a real call processing would generate: First the number of one service customer is chosen by random. Then the service customer and the associated real telephone numbers are retrieved. One of these numbers is randomly selected as routing destination. At the end of the call processing one new record will be inserted into the `CALLS` table. Furthermore a calculation of the sales volume of our service provider and calculations of the charges for our service customers are included in our example workload too:

$$Workload = Call\ Processing + Charge\ Calculation + Sales\ Volume\ Calculation$$

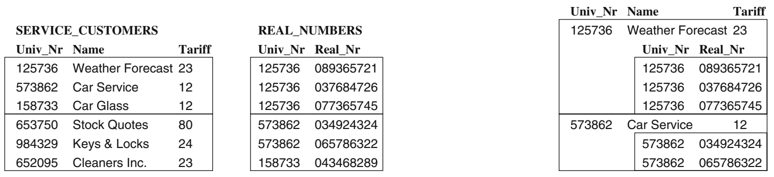
This workload comprises several access patterns: To retrieve a service customer record during the call processing an index is used. For the calculation of the sales volume of the service provider and for the calculations of the charges for the service customers sequential scans are performed. This way we reached near realistic access schemes.

4.1 Alternative Clustering Strategies

To store the data of service customers and their real telephone numbers we considered the following two clustering strategies.

Object-Spanning Clustering. First we stored the data fully normalized as usual in relational databases (Figure 9). We created the tables `SERVICE_CUSTOMERS`, `REAL_NUMBERS` and `CALLS` with the usual primary and foreign key indexes.

The following database accesses have to be performed per simulated call:



SERVICE_CUSTOMERS

Univ_Nr	Name	Tariff
125736	Weather Forecast	23
573862	Car Service	12
158733	Car Glass	12
653750	Stock Quotes	80
984329	Keys & Locks	24
652095	Cleaners Inc.	23

REAL_NUMBERS

Univ_Nr	Real_Nr
125736	089365721
125736	037684726
125736	077365745
573862	034924324
573862	065786322
158733	043468289

Object-spanning clustering

Univ_Nr	Name	Tariff
125736	Weather Forecast	23
	Univ_Nr	Real_Nr
	125736	089365721
	125736	037684726
	125736	077365745
573862	Car Service	12
	573862	034924324
	573862	065786322

Object-based clustering

Fig. 9. Cluster strategies

– One read access to SERVICE_CUSTOMERS to check whether the service is available and to retrieve the tariff for the service user using the primary index on UNIV_NUMBER.
– Read accesses to REAL_NUMBERS using the index on UNIV_NUMBER.
– Write accesses to CALLS and the associated indexes to record the call.

To calculate the charges, the table SERVICE_CUSTOMERS has to be scanned completely and all associated records from CALLS have to be fetched using the foreign key index on UNIV_NUMBER. To calculate the sales volume, the table CALLS has to be scanned summing up the charges.

Object-Based Clustering. In the second example we physically clustered the tables with the service customers and the real telephone numbers (Figure 9). This is a good simulation of the object-based clustering strategy. A B-Tree index realizes the access path on the common attribute UNIV_NUMBER. The additional inclusion of calls into the cluster would be consequently object-oriented and needed for the full object-based clustering like described in Section 2.4. But inserting large numbers of calls rises the risk of degeneration in the storage structures like overflow records with decreasing performance. So we used a compromise and separated the call data from the service customer.

In this case the following accesses are needed to process a telephone call:

– One read access to the cluster with the called service customer including the real telephone numbers using the index on UNIV_NUMBER. It is not necessary to use another index as in the first case because the real numbers are stored in the same cluster and can be accessed directly.
– Accesses to the table CALLS and its indexes as in the first case.

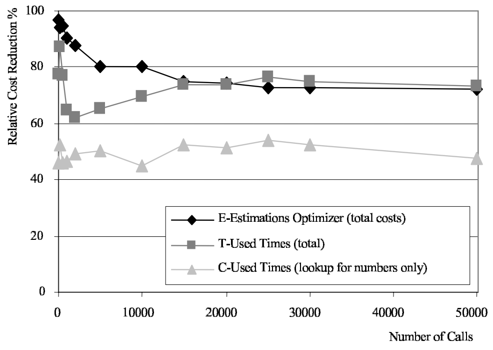
To calculate the charges, all clusters with service customers and real telephone numbers have to be scanned completely. Additionally for each service customer all associated call records are fetched using the index on UNIV_NUMBER. To calculate the sales volume, the table CALLS has to be scanned as in the first case.

4.2 Comparison and Discussion

We assume that each variant of clustering supports a manner of processing well. Object-based clustering supports the call processing best because all necessary data is stored clustered and only one index is needed. But the calculation of the charges for each service customer is expensive because the read data blocks contain unnecessary records with real telephone numbers in addition to the needed service customer records.

Using object-spanning clustering it is more complex to retrieve the data needed for call processing because the data of the service customers and their real telephone numbers are divided into two tables. But the calculation of the charges is less expensive.

The decision for one of the variants depends on the ratio of data accesses for the call processing on the one hand and sequential scans for the calculation of



$$E = EC/ED$$
$$T = TC/TD$$
$$C = CC/CD$$

Fig. 10. Cost reduction by object-based clustering

charges on the other hand. Therefore we changed this ratio in our experimental workload by gradually increasing the number of processed calls. For each workload we collected the following values:

– The estimation of the total cost to process the workload provided by the query optimizer using object-spanning clustering (ED) and using object-based clustering (EC).
– The time the simulation program needs to process the workload using object-spanning clustering (TD) and using object-based clustering (TC).
– The time the simulation program needs to retrieve only the data for the call processing using object-spanning clustering (CD) and using object-based clustering (CC).

Then we calculated the quotients E, T and C to obtain relations between object-based and object-spanning clustering. The used formulas and the results are shown in Figure 10. The diagram shows that in our example the estimated and the measured ratio swings into approximately 75%. This means a cost reduction of nearly 25% is reachable by changing the clustering strategy. The curves in the diagram show that the benefits of object-based clustering in our example rise with the quota of call processing. This is because the call processing profits from object-based clustering, as we assumed.

The diagram in Figure 10 also shows a high degree of consistency of the estimated and the measured values. It shows that the cost reduction depends on the operation mix in the workload. Therefore it is important to have a realistic evaluation of the workload.

The main benefit of our method is that the workload and the cost estimations could be derived from the real production system taking into account the systems environment and configuration without impairing it. Furthermore the results are not based on any assumptions.

5 Conclusion and Outlook

The paper presented an approach for the optimization of physical storage structures of complex nested objects in ORDBMS. Our approach bases mainly on two concepts: The first is a newly developed language for the specification of physical storage structures called PRDL. The second is a methodological framework for the assessment of different storage structures based on workload capturing.

The design of PRDL is based on a classification of possible physical storage structures. The language provides an interface to the physical object mapping in ORDBMS and allows to specify storage structures explicitly. The language is designed to be independent from SQL-DDL and helps to enhance data independency and the separation between logical and physical design.

The methodological framework for the assessment of storage structures comprises a method that captures a typical database workload in a real production environment. Using the DBMS optimizer it permits cost estimations under realistic circumstances and allows to assess alternative storage variants with respect to the captured workload. A major advantage of our method is that it avoids expensive data restructuring using trial and error.

Within the scope of the Jena IBM SUR Grant project “Object-Relational Database Features and Extensions: Model and Physical Aspects” research activities include the following: SQL:1999 is extended by fully-fledged collection support, it is investigated into the implementation of these extended SQL constructs with special focus on physical storage structures. A prototype intended to serve as test bed for implementation alternatives, performance tests and usability studies is currently being developed. Future work includes the analysis of storage structures adapted to specific workloads. And furthermore a simulation environment is constructed allowing storage experiments on a physical record level.

References

1. Stonebraker, M.: Inclusion of new types in relational data base systems. In: Proceedings of the Second International Conference on Data Engineering, February 5–7, 1986, Los Angeles, California, USA, IEEE Computer Society (1986) 262–269
2. Stonebraker, M., Moore, D.: Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann (1996)
3. Lorie, R.A., Kim, W., McNabb, D., Plouffe, W., Meier, A.: Supporting complex objects in a relational system for engineering databases. In Kim, W., Reiner, D.S., Batory, D.S., eds.: Query Processing in Database Systems. Springer (1985) 145–155
4. Wilkes, W., Klahold, P., Schlageter, G.: Complex and composite objects in CAD/CAM databases. In: Proceedings of the Fifth International Conference on Data Engineering, February 6–10, 1989, Los Angeles, California, USA, IEEE Computer Society (1989) 443–450
5. Navathe, S.B., Cornelio, A.: Modeling physical systems by complex structural objects and complex functional objects. In Bancilhon, F., Thanos, C., Tsichritzis, D., eds.: Advances in Database Technology – EDBT’90. International Conference on Extending Database Technology, Venice, Italy, March 26–30, 1990, Proceedings. Volume 416 of Lecture Notes in Computer Science., Springer (1990) 238–252

6. ISO: ISO/IEC 9075-1:1999, Information technology – Database languages – SQL. (1999)
7. Carey, M.J., Chamberlin, D.D., Narayanan, S., Vance, B., Doole, D., Rielau, S., Swagerman, R., Mattos, N.M.: O-o, what have they done to DB2? In Atkinson, M.P., Orlowska, M.E., Valduriez, P., Zdonik, S.B., Brodie, M.L., eds.: VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7–10, 1999, Edinburgh, Scotland, UK, Morgan Kaufmann (1999) 542–553
8. Krishnamurthy, V., Banerjee, S., Nori, A.: Bringing object-relational technology to mainstream. In Delis, A., Faloutsos, C., Ghandeharizadeh, S., eds.: SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1–3, 1999, Philadephia, Pennsylvania, USA, ACM Press (1999) 513–514
9. Brown, P.: Implementing the spirit of SQL-99. In Delis, A., Faloutsos, C., Ghandeharizadeh, S., eds.: SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1–3, 1999, Philadephia, Pennsylvania, USA, ACM Press (1999) 515–518
10. Lufter, J.: Datentypkonzepte und funktionaler Vergleich einiger objektrelationaler Datenbanksysteme. Jenaer Schriften zur Mathematik und Informatik Math/Inf/99/02, Institut für Informatik, Friedrich-Schiller-Universität Jena (1999) (in German).
11. Kauhaus, C., Lufter, J., Skatulla, S.: Eine Transformationsschicht zur Realisierung objektrelationaler Datenbankkonzepte mit erweiterter Kollektionsunterstützung. Datenbank-Spektrum **2** (2002) 49–58 (in German).
12. Zaniolo, C.: The representation and deductive retrieval of complex objects. In Pirotte, A., Vassiliou, Y., eds.: VLDB'85, Proceedings of 11th International Conference on Very Large Data Bases, August 21–23, 1985, Stockholm, Sweden, Morgan Kaufmann (1985) 458–469
13. Copeland, G.P., Khoshafian, S.: A decomposition storage model. In Navathe, S.B., ed.: Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28–31, 1985, ACM Press (1985) 268–279
14. Jhingran, A., Stonebraker, M.: Alternatives in complex object representation: A performance perspective. In: Proceedings of the Sixth International Conference on Data Engineering, February 5–9, 1990, Los Angeles, California, USA, IEEE Computer Society (1990) 94–102
15. Kessler, U., Dadam, P.: Benutzergesteuerte, flexible Speicherungsstrukturen für komplexe Objekte. In: BTW 1993. (1993) 206–225 (in German).
16. Skatulla, S.: Storage of complex types with collection-valued attributes in object-relational database systems. In: Tagungsband des 14. Workshop Grundlagen von Datenbanken, Fischland/Darss, Mai 2002. Preprint CS-01-02, Universität Rostock (2002) 106–111
17. Kissel, F.: Physische Speicherung komplexer Objekte mit kollektionswertigen Attributen in ORDBMS. Studienarbeit, Friedrich-Schiller-Universität Jena (2002) (in German).
18. Härder, T., Reuther, A.: Concepts for implementing a centralized database management system. In: Proc. Int. Computing Symposium on Application Systems Development, Teubner (1983) 28–59
19. Markl, V., Ramsak, F., Bayer, R.: Improving OLAP performance by multidimensional hierarchical clustering. In: IDEAS'99, Proceedings of IDEAS Conference, Montreal, Canada, 1999. (1999) 165–177

20. Oracle Corporation, Redwood City, CA: Oracle9i Database Concepts. (2001)
21. Saake, G., Heuer, A.: Datenbanken: Implementierungstechniken. MITP (1999) (in German).
22. Härder, T., Rahm, E.: Datenbanksysteme, Konzepte und Techniken der Implementierung. Springer (2001) (in German).
23. Bell, S.: Semantische Anfrageoptimierung. Informatik-Spektrum **19** (1996) (in German).
24. Chaudhuri, S., Gupta, A.K., Narasayya, V.: Compressing sql workloads. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, ACM Press (2002) 488–499
25. Chaudhuri, S., Narasayya, V.R.: Autoadmin 'what-if' index analysis utility. In Haas, L.M., Tiwary, A., eds.: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2–4, 1998, Seattle, Washington, USA, ACM Press (1998) 367–378

Hierarchical Bitmap Index: An Efficient and Scalable Indexing Technique for Set-Valued Attributes*

Mikołaj Morzy[1], Tadeusz Morzy[1], Alexandros Nanopoulos[2], and Yannis Manolopoulos[2]

[1] Institute of Computing Science
Poznań University of Technology, Piotrowo 3A, 60-965
Poznań, Poland
{Mikolaj.Morzy,Tadeusz.Morzy}@cs.put.poznan.pl
[2] Department of Informatics, Aristotle University of Thessaloniki
Thessaloniki, Greece
{alex,manolopo}@delab.csd.auth.gr

Abstract. Set-valued attributes are convenient to model complex objects occurring in the real world. Currently available database systems support the storage of set-valued attributes in relational tables but contain no primitives to query them efficiently. Queries involving set-valued attributes either perform full scans of the source data or make multiple passes over single-value indexes to reduce the number of retrieved tuples. Existing techniques for indexing set-valued attributes (e.g., inverted files, signature indexes or RD-trees) are not efficient enough to support fast access of set-valued data in very large databases.
In this paper we present the hierarchical bitmap index—a novel technique for indexing set-valued attributes. Our index permits to index sets of arbitrary length and its performance is not affected by the size of the indexed domain. The hierarchical bitmap index efficiently supports different classes of queries, including subset, superset and similarity queries. Our experiments show that the hierarchical bitmap index outperforms other set indexing techniques significantly.

1 Introduction

Many complex real world objects can be easily modeled using set-valued attributes. Such attributes appear in several application domains, e.g., in retail databases they can represent the set of products purchased by a customer, in multimedia databases they can be used to represent the set of objects contained in an image, in web server logs they correspond to web pages and links visited by a user. Finally, in data mining applications set-valued attributes are commonly used to store time-series and market basket data. Contemporary database systems provide the means to store set-valued attributes in the database (e.g., as

* Work supported by a Bilateral Greek-Polish Program

contents of nested tables or values of user-defined types), but they don't provide either language primitives or indexes to process and query such attributes.

The inclusion of set-valued attributes in the SQL3 standard will definitely result in wide use of such attributes in various applications. Web-based interfaces to database systems open new possibilities but require prompt advances in query processing on sets. Examples of advanced applications making heavy use of set-valued attributes are, among others, automatic recommendations and discount offers, collaborative filtering, or web site personalization.

Another domain that would greatly profit from the possibility to perform advanced querying on sets is data mining. Several knowledge discovery techniques rely on excessive set processing. Shifting these computations directly to the database engine would result in considerable time savings. This can be observed especially in case of the apriori family of algorithms which perform huge number of subset searches. In the apriori framework the frequency of different sets of products is determined during repeated scans of the entire database of customer transactions. In each pass every transaction is tested for the containment of the so-called candidate sets, the sets of products that the algorithm suspects to appear frequently enough in the database. These repetitive tests are in fact subset queries. Presently, due to the lack of support from the commercial database management systems, these tests are conducted on the application side and not in the database.

Developing efficient mechanisms for set indexing and searching can serve as the basis for other improvements. The ability to retrieve sets based on their subset properties can lead to the improvement of many other algorithms that depend on database set processing. As an example let us consider a system for automated web site personalization and recommendation. Such systems are commonly used in various e-commerce sites and on-line stores. The system keeps information about all products purchased by the customers. Based on these data the system discovers patterns of customer behaviour, e.g., in the form of characteristic sets of products that are frequently purchased together. When a new potential customer browses the on-line catalog of available products, the system can anticipate which product groups are interesting to a customer by analyzing search criteria issued by a customer, checking visited pages and inspecting the history of purchases made by that customer, if the history is available. By issuing subset queries to find patterns relevant to the given customer the system can not only dynamically propose interesting products, preferably at a discount, but it can also tailor the web site navigation structure to satisfy specific customers requirements.

Let us assume that the customer started from the main page of an on-line store and issued a search with a key word "palm". Furthermore, let us assume that the identification of the customer is possible, e.g., by the means of a cookie or a registered username. The system retrieves then the history of all purchases made by that specific customer. In the next step, the system uses a subset query to find all patterns pertaining to palm devices. Another query is required to find those patterns that are eligible for the given customer. This is done by searching for palm related patterns that are subsets of the history of customer purchases. Those patterns are used to predict future purchases of the customer. On the other hand, the system could also store web site navigation patterns discovered from web server logs. Such patterns tend to predict future navigation paths based on previously visited pages. The system could use a subset query to find those patterns that are applicable to the customer and determine that this specific customer is interested mainly in computers. The system could hide the uninteresting departments of the on-line store by shifting links to those departments to less accesible parts of the page and moving forward links to palm related products (software, accessories, etc.) to ease navigation. This scenario, which could be used in e-commerce sites, assumes that the subset searches are performed very efficiently and transparently to the customer. This assumption represents the fact that the entire processing must finish within a very short time-window in order not to let the customer become annoyed with the web site response time and not to let him move to another on-line store.

Unfortunately, currently available database systems do not provide mechanisms to achieve satisfactory response times for subset queries and database sizes in question. The ability to efficiently perform set-oriented queries in large data volumes could greatly enhance web-based applications. This functionality is impatiently anticipated by many potential users.

The most common class of queries which often appear in terms of set-valued attributes are subset queries that look for sets that entirely contain a given subset of elements. Depending on the domain of use subset queries can be used to find customers who bought specific products, to discover users who visited a set of related web pages, to identify emails concerning a given matter based on a set of key words, and so on. In our study and experiments we concentrate on subset queries as they are by far the most common and useful class of queries regarding set-valued attributes. We recognize other classes of queries as well and we describe them later on.

To illustrate the notion of a subset query let us consider a small example. Given a sample database of retail related data consisting of a relation `Purchases(trans_id,cust_id,products)`. An example of a subset query is:

```
SELECT COUNT(DISTINCT cust_id)
FROM Purchases
WHERE products ⊇ {‘milk’,‘butter’};
```

This query retrieves the number of customers who purchased products *milk* and *butter* together during a single visit to the supermarket. Such queries require costly set inclusion tests performed on very large data volumes. There could be as many as tens of thousands of customers per day, each purchasing several products. The number of different products in an average supermarket could easily amount to hundreds of thousands. Standard SQL language doesn’t contain primitives to formulate such queries. Usually, those queries are written in an awkward and illegible way. For example, the aforementioned subset search query can be expressed in standard SQL in one of the following manners.

```
SELECT COUNT(DISTINCT A.cust_id)
FROM Purchases A, Purchases B
WHERE A.trans_id = B.trans_id
AND A.item = ‘milk’ AND B.item = ‘butter’;
```

or

```
SELECT COUNT(*) FROM (
   SELECT trans_id FROM Purchases
   WHERE item IN {‘milk’,‘butter’}
   GROUP BY trans_id
   HAVING COUNT(*) = 2 );
```

Both queries are very expensive in terms of the CPU and memory utilization and tend to be time-consuming. It is easy to notice that the number of self joins in the first query is proportional to the size of the searched set. The second query requires grouping of very large table, which may be very costly, too. Additionaly, both queries are cryptic, hard to read, and lack flexibility. Adding another element to the searched set requires in the first case modifying the `FROM` clause and adding two new predicates; in the second case it requires modifying the `WHERE` and `HAVING` clauses.

Typical solution to speed up queries in database systems is to use indexes. Unfortunately, despite the fact that the SQL3 supports set-values attributes and most commercial database management systems offer such attributes, no commercial DBMS provides to date indexes on set-valued attributes. The development of an index for set-valued attributes would be very useful and would improve significantly the performance of all applications which depend on set processing in the database.

Until now, several indexing techniques for set-valued attributes have been proposed. These include, among others, R-trees [10], signature files [4] and S-trees [6], RD-trees [11], hash group bitmap indexes [14], or inverted files [2]. Unfortunately, all those indexes exibit deficiencies that limit their use in real-world applications. Their main weaknesses include the non-unique representation of the indexed sets, the necessity to verify all hits returned from the index in order to prune false hits, and significant losses in performance when the cardinality of the indexed sets grows beyond a certain threshold. Another drawback of the aforementioned methods is the fact that most of them, excluding S-trees, can’t be used to speed up queries other than subset queries.

In this paper we present the hierarchical bitmap index. It is a novel structure for indexing sets with arbitrary sized domains. The main advantages of the hierarchical bitmap index are:

- scalability with regard to the size of the indexed domain and to the average size of the indexed set,
- efficient support of different classes of queries, including subset, superset and similarity queries,
- compactness which guarantees that the index consumes the least amount of memory required,
- unique representation of the indexed sets which attains the lack of any false hits and avoids the cost of false hit resolution.

We prove experimentally that the hierarchical bitmap index outperforms other indexes significantly under all circumstances. In the next sections we describe the structure of the index in detail.

Our paper is organized as follows. We begin in Section 2 with the presentation of previously proposed solutions. Section 3 contains the description of the hierarchical bitmap index. Algorithm for performing subset queries using our index is given in Section 4. We present the results of the conducted experiments in Section 5. Section 6 contains the description of other classes of set-oriented queries, namely the superset and similarity queries, that can be efficiently processed using the hierarchical bitmap index. We also provide algorithms to process these classes of queries using the hierarchical bitmap index. Finally, we conclude in Section 7 with a brief summary and the future work agenda.

2 Related Work

Traditional database systems provide several indexing techniques that support single tuple access based on atomic attribute value. Most popular indexes include B+ trees [5], bitmap indexes [3] and R-trees [10]. However, these traditional indexes do not provide mechanisms to efficiently query attributes containing sets. Indexing of set-valued attributes was seldom researched and resulted in few proposals.

First access methods were developed in the area of text retrieval and processing. One of the techniques proposed initially for indexing text documents is inverted file [2]. Inverted file consists of vocabulary and occurrences. Vocabulary is the list of all elements that appear in the indexed sets. Identifiers of all sets containing given element are put on a list associated with that element. All lists combined together form the occurrences. When a user searches for sets containing a given subset, the occurrence lists of all elements belonging to the searched subset are retrieved and joined together to determine the identifiers of sets appearing on all occurrence lists. This technique can be successfully used when indexing words occurring in documents. The number of lists is relatively low (usually only the key words are indexed) and the occurrence lists are short. Unfortunately, in retail databases the size of the vocabulary and the number of occurrence lists are huge. For this reason inverted files are not well suited to index market basket data. Another disadvantage of the inverted file is the fact that this kind of index can't be used to answer superset or similarity queries.

Signature files [4] and signature trees (S-trees) [6] utilize the superimposition of set elements. Each element occurring in the indexed set is represented by a bit vector of the length n with k bits set to '1'. This bit vector is called the *signature* of the element. Signatures are superimposed by a bitwise `OR` operation to create a set signature. All set signatures can be stored in a sequential signature file [7], a signature tree, or using an extendible signature hashing. Interesting performance evaluation of signature indexes can be found in [12]. Much effort is put into further improving signature indexes [17]. Signature indexes are well suited to handle large collections of sets but their performance degrades quickly with the increase of the domain size of the indexed attribute. Every element occurring in a set must be represented by a unique signature. Large number of different elements leads to either very long signatures or to hashing. Long signatures are awkward to process and consume large amounts of memory. On the other hand, hashing introduces ambiguity of mapping between elements and signatures, increasing the number of false hits. In case of very large databases this could lead to unacceptable performance loss.

Indexing of set-valued attributes was also researched in the domain of object-oriented and spatial database systems [16]. Evaluation of signature files in object-oriented database environment can be found in [13]. Processing of spatial data resulted in the development of numerous index types, among them R-trees [10] and BANG indexes [8]. These indexes can also be used to index non-spatial objects, provided there is a way to map an indexed object to a multidimensional space. In case of market basket data each product appearing in indexed sets must be represented as a distinct dimension with dimension members 0 (product not present in a given basket) and 1 (product present in a given basket). However, the number of possible dimensions involved in indexing set-valued attributes, which equals tens of thousands of dimensions, makes this approach unfeasible. A modification of the R-tree index, called the Russian Doll tree (RD-tree), was presented in [11]. In the RD-tree all indexed sets are stored in tree leaves, while inner nodes contain descriptions of sets from the lower levels of the tree. Every inner node fully describes all its child nodes placed on lower levels. RD-trees can be used only to answer subset queries and their performance drops significantly with the increase of the database size.

Subset search is crucial in many data mining activities. From the knowledge discovery domain originates another proposal, namely the hash group bitmap index [14]. This index is very similar to signature index. Every element appearing in the indexed set is hashed to a position in the bit vector. For large domains there can be several elements hashing to the same position. Set representations are created by the superimposition of bits representing elements contained in the given set. Subset search is performed in two phases. First, in the filtering step, the index is scanned to find matching entries (these are all sets that potentially contain the given subset). Then, in the verification step, all matching entries are tested for the actual containment of the searched set to eliminate any false hits. For very large domains the ambiguity introduced by hashing results in rapid increase of the number of false hits, which directly affects the performance of the index.

3 Hierarchical Bitmap Index

Hierarchical bitmap index is based on signature index framework. It employs the idea of exact set element representation and uses hierarchical structure to compact resulting signature and reduce its sparseness. The index on a given attribute consists of a set of index keys, each representing a single set. Every index key comprises a very long signature divided into n-bit chunks (called *index key* *leaves*) and a set of *inner nodes* of the index key organized into a tree structure. The highest inner node is called the *root* of the index key. The signature must be long enough to represent all possible elements appearing in the indexed set (usually hundreds of thousands of bits). Every element o_i of the attribute domain A, $o_i \in dom(A)$ is mapped to an integer i.

Given an indexed set $S = \{o_1, o_2, \ldots, o_n\}$. The set is represented in the index in the following way. Let l denote the length of the index key node. An element $o_i \in S$ is represented by a '1' on the j-th position in the k-th index key leaf, where $k = \lceil i/l \rceil$ and $j = i - (\lceil i/l \rceil - 1) * l$. Therefore, the set S is represented by n '1's set at appropriate positions of the index key leaves. An index key node (either leaf node or inner node) which contains '1' on at least one position is called the *non-empty* node, while an index key node which contains '0' on all positions is called the *empty* node. The next level of the index key compresses the signature representing the set S by storing information only about the non-empty leaf nodes. A single bit in an inner node represents a single index key leaf. If this bit is set to '1' then the corresponding index key leaf contains at least one position set to '1'. The i-th index key leaf is represented by j-th position in the k-th inner index key node, where $k = \lceil i/l \rceil$ and $j = i - (\lceil i/l \rceil - 1) * l$. Every upper level of the inner nodes represents the lower level in an analogous way. This procedure repeats recursively to the index key root. The index key stores only the non-empty nodes. Empty nodes are not stored anywhere in the index key.

The two parameters which affect the shape and the capacity of the index are: l – the length of a single index key node and d – the depth of the index key tree structure. For example, if $d = 4$ and $l = 32$ then the root of the index key contains 32 positions, the second level contains $32^2 = 1024$ positions and the third level contains $32^3 = 32768$ positions that correspond to 32768 index key leaves. This allows storage in a single index key with $d = 4$ and $l = 32$ information about a set that could contain at most $32^4 = 1048576$ different elements. Furthermore, in case this size is not sufficient, extending the index key to five levels would allow indexing sets with the domain of 33554432 different elements.

The presented index has several advantages over previously presented approaches. The most important feature of the hierarchical bitmap is the fact that it supports indexing sets with no loss of accuracy and no ambiguity. Every indexed set is represented in the index uniquely. The index permits representing sets of arbitrary length drawn from a domain of arbitrary size as long as the size of the domain doesn't exceed the maximum cardinality. Note, from the numbers given above, that the index with only four levels allows indexing sets with the domain of 1048576 different elements. This size is sufficient for most domains in practical applications.

Example 1. Assume index key node length $l = 4$ and index depth $d = 3$. Assume also that the elements of the attribute domain have been already mapped to integer values. Given the set $S = \{2, 3, 9, 12, 13, 14, 38, 40\}$. The index key of the set S is depicted in Fig. 1. At the lowest level (level 3) of the index 8 bits

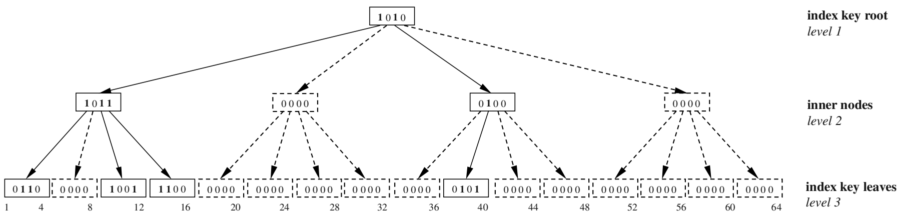


Fig. 1. A single key of the hierarchical bitmap index

corresponding to the given elements of the set S are set to '1'. As a result, index key leaf nodes 1,3,4 and 10 become non-empty (they are marked with a solid line). At the upper level (level 2) 4 bits representing non-empty leaf nodes are set to '1'. In the root of the index key only first and third bits are set to '1', which means that only first and third inner nodes at the level 2 are non-empty. Notice that the index consists of only 4 index key leaf nodes, 2 inner index key nodes at the level 2 and a single index key root. Empty nodes (marked with a dotted line) are not stored anywhere in the index and are shown in the figure to better explain the idea of the hierarchical bitmap index. Note that the index key node size has been set to 4 bits for demonstration purposes only. In real world applications it would be most likely set to one machine word, i.e., to 32 bits.

□

Let us now discuss briefly the physical implementation of the hierarchical bitmap index. An example of the entire index is depicted in Fig. 2. Every index key is stored as a linked list of all index key nodes (both internal and leaf nodes) except the index key root. Those linked lists are stored physically in a set of files, where each file stores all index keys containing an equal number of nodes (e.g., file 1 contains index keys composed of five nodes and file 2 contains index keys composed of 2 nodes). As the result, every file contains only the records of a fixed size. This allows efficient maintenance procedures, e.g., inserting a new index key requires counting the number of nodes in the key and appending a fixed sized record at the end of the appropriate file. To store the index key roots we propose a hybrid S-tree in which all tree leaves are connected with pointers to form a list. In this fashion the index can be efficiently processed by both S-tree traversal in case of subset queries and full scanned in case of superset and similarity queries. The leaves of the S-tree contain all index key roots and, for every index key root, the pointers to the locations where the rest of a given index key is stored. Internal nodes of the S-tree contain the superimposition of all index key roots contained in the lower level nodes that they point to. An interesting discussion on signature trees construction and improvement can be found in [17]. We believe that it is sufficient to store in the S-tree only the roots of the index keys. This allows early pruning of sets that don't contain the searched subset while it keeps the S-tree compact and easy to manage. This architecture makes update operations become more costly because updating a given key might require relocating the

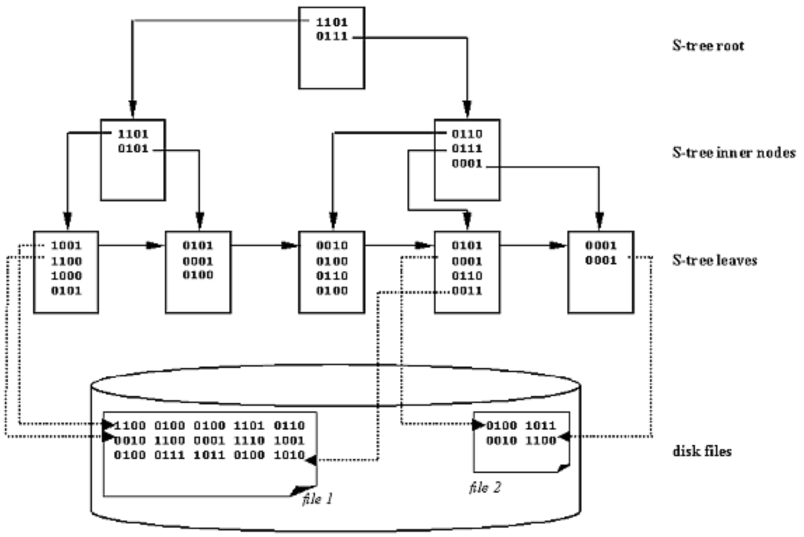


Fig. 2. S-tree with hierarchical bitmap index key roots

key to another file (if the number of nodes changed after the update). On the other hand, we argue that the update operations are not frequent enough to result in a noticeable preformance loss.

4 Search Algorithm

Hierarchical bitmap index can be used to speed up different classes of queries involving set-valued attributes. Given a relation $R = \{t_1, t_2, \ldots, t_n\}$ of tuples containing a set-valued attribute A. Let $\{A_1, A_2, \ldots, A_n\}$ denote the values of the attribute A in subsequent tuples of R, $t_i.A = A_i$. A query predicate is defined by the set-valued attribute A and a set comparison operator $\Theta \in \{\subseteq, \supseteq, \approx\}$. Let q denote a finite set of elements of the attribute A searched by a user. We will further refer to q as the user's query.

Assume the hierarchical bitmap index on the attribute A of the relation R is defined. Let $K(A_i)$ denote the index key of the set A_i. Let $N_n^m(A_i)$ denote the n-th node (either internal or leaf) at the m-th level of the index key of the set A_i. Let $N_1^1(A_i)$ denote the root of the index key of A_i. Let d denote the depth of the index key and & denote the bitwise `AND` operation.

The goal of the search algorithm is to find all tuples $t_i \in R$, such that $t_i.A \supseteq q$, given the search set q. The search algorithm is presented below.

```
1: for all $A_i \in R.A$ from the S-tree such that $N_1^1(A_i) \& N_1^1(q) = N_1^1(q)$ do
2:    for all levels $l$ do
3:       for all internal nodes $i$ at level $l$ in $q$ do
```

```
 4:         $p = \texttt{skip}(A_i, l, i)$
 5:         if $N^l_{p+1}(A_i) \& N^l_i(q) \neq N^l_i(q)$ then
 6:            return(false)
 7:         end if
 8:      end for
 9:   end for
10:   return(true)
11: end for
```

The algorithm starts with defining the index key on the searched set q. Then, the algorithm traverses the S-tree to find all sets that potentially contain the searched subset. This is determined by comparing index key roots of each set A_i with the index key root of the searched set q.

For each set A_i found in the S-tree, the algorithm begins from the root of the index key of the searched set q and recursively tests all nodes at all levels of the index key of the searched set q comparing them with the corresponding nodes in the index key of A_i. Both index keys may contain different number of nodes because there can exist non-empty nodes in the index key of A_i that are not relevant to the query. These non-relevant nodes correspond to index key leaf nodes containing elements that are not present in the query. Thus, the main difficulty in comparing index keys is to determine pairs of corresponding nodes in compared index keys. To cope with this problem we introduce the function $\texttt{skip}(A_i, l, i)$ (line 4). The funcion $\texttt{skip}(A_i, l, i)$ computes at the level l the number of nodes that have to be skipped in the index key of A_i to reach the node corresponding to $N^l_i(q)$.

The function computes the number of nodes to be skipped on the basis of the parent node at the higher level. Given internal node i at the level l of the index key of q. The function retrieves the node $N^{l-1}_{i\%d+1}(A_i)$ (where % denotes the modulo operator; this is the parent of the i-th node). The number of nodes that must be skipped in A_i is equal to the number of positions in $N^{l-1}_{i\%d+1}(A_i)$ set to '1' and preceding the position $(i\%d) + 1$. The test for the containment of corresponding nodes is performed in the line (5).

As can be easily noticed, the algorithm requires, for index key roots retrieved from the S-tree, accessing of some part of the linked list of non-empty nodes. This access introduces some overhead. Nevertheless, this additional cost of the linked list traversal is negligible when compared to the cost of veryfying every hit and resolving false hits as required by all other indexing techniques. Besides, the average cost of the linked list is relatively low because many index keys are rejected very early. The hierarchical bitmap index performs significant pruning at the upper level of the index keys. The degree of pruning depends on the mapping of set elements to positions in the signature at the lowest level of the index. We intend to investigate this dependency in greater detail in future experiments. The ability to prune index keys at the upper levels combined with the lack of necessity to verify false hits provides the biggest pay-off in comparison with other set-oriented indexes.

Example 2. Consider the index key of the set S from the Example 1. It is stored in the form of the linked list $K(S) = \langle 1010, 1011, 0100, 0110, 1001, 1100, 0101 \rangle$. Let user query contain elements $q = \{9, 13, 40\}$. The algorithm starts with defining the index key on the searched set q. The index key for the query q is $K(q) = \langle 1010, 0011, 0100, 1000, 1000, 0001 \rangle$. Then, the algorithm begins at the first level and tests whether $N_1^1(S) \& N_1^1(q) = N_1^1(q)$. This yelds **true** because $1010 \& 1010 = 1010$. Next, the algorithm moves to the next level and considers the node $N_2^1(q) = 0011$. Function $\texttt{skip}(S, 2, 1)$ returns 0 as there are no nodes to be skipped in $K(S)$. Comparing $N_2^1(S)$ and $N_2^1(q)$ returns $1011 \& 0011 = 0011$ which is true. Test on $N_2^2(S)$ and $N_2^2(q)$ also succeeds. Then the algorithm advances to the third level. Function $\texttt{skip}(S, 3, 1)$ returns 1 because the node $N_3^1(S) = 0110$ must be skipped as it represents the items $\{2, 3\}$ which are not relevant to the query. Thus the next test compares nodes $N_3^2(S) = 1001$ and $N_3^1(q) = 1000$. This procedure continues until there are no more non-empty nodes in the index key of q. It is easy to notice that the remaining comparisons will succeed – $N_3^3(S) = 1100$ contains $N_3^2(q) = 1000$ and $N_3^4(S) = 0101$ also contains $N_3^3(q) = 0001$. Therefore, we conclude that the set S contains the searched set q.

□

The algorithm is very robust. Most operations are bit manipulations (bitwise `AND`) that perform very quickly. The most important feature of the algorithm is the fact that the hierarchical bitmap index always returns an exact answer and never generates any false hits. Therefore, the search algorithm doesn't contain any verification steps which are required when using other set indexing techniques. The elimination of the verification phase and the lack of the false hits is the main reason for which hierarchical bitmap index is superior to other set indexes and outperforms them significantly. In the next section we will present the results of the experimental evaluation of the index.

5 Experimental Results

The experiments were conducted on top of the Oracle 8i database management system running under Linux with two Pentium II 450 MHz processors and 512 MB memory. Data sets were created using DBGen generator from the Quest Project [1]. Table 1 summarizes the values of different parameters that affect the characteristics of the data sets used in our experiments. Data sets tend to mimic typical customer transactions in a supermarket. Number of distinct elements varies from relatively small (1000) to rather large (30000) (this is the number of different products being sold in a typical mid-sized supermarket). The size of a single set (40 elements) also represents the real size of the average customer basket in a supermarket. Query sizes vary from general queries (considering one or two elements) to very specific (considering several elements). The number of indexed sets varies from 1000 (very small database) to 100000 (relatively big database). In our experiments we compare the hierarchical bitmap index with two similar techniques, namely the signature index and the hash index.

Table 1. Synthetic data parameters

parameter	value
number of itemsets	1000 to 250000
domain size	1000 to 30000
itemset size	40
query size	1 to 20

Figure 3 presents the average response time for different searched set sizes. The response time is the average response time computed for the database size varying from 1000 sets to 250000 sets and the size of the domain of the indexed attribute varying from 1000 to 30000 elements. All subsequent response times are also computed as an average over the entire parameter range in Tab 1. The most interesting about the hierarchical bitmap index is the fact that the response time remains constant. Very general queries using small searched sets (1–5 elements) exhibit the same response times as very specific queries using large searched sets (10–20 elements). For general queries the hierarchical bitmap index outperforms other indexes significantly, which can be explained by the fact that the hierarchical bitmap index doesn't produce any false hits and doesn't need any verification. On the other hand, both signature and hash indexes must verify every set returned from the index. For general queries we expect a lot of ambiguity in signature and hash indexes. Therefore, the answers are several times slower. We argue that these general queries are most frequent in real world applications and that this fact even stronger advocates the use of our index. Nevertheless, even for very specific queries the hierarchical bitmap index performs two times better than the other techniques.

The results of the next experiment are depicted in Fig. 4. It presents the average response time with respect to the varying number of elements in the domain of the indexed attribute. It can be observed that for all domain sizes the hierarchical bitmap index performs several times better than signature or hash indexes. Again, it is worth noticing that the response time remains almost constant for all domain sizes. For small domains (1000 elements) our index is 10–15 times faster than the other techniques. For larger domains the difference diminishes but remains substantial.

Figure 5 shows the scalability of the hierarchical bitmap index in comparison with the remaining two indexes with regard to the database size. The worst performance can be observed with the hash index, because the increase in the database size results in more hits that have to be verified. Signature index produces less false hits but all answers still have to be verified. Again, the biggest gain of the hierarchical bitmap index lies in the fact that it doesn't produce any false hits and doesn't require any verification step. The response time of the hierarchical bitmap index scales linearly with the database size.

Finally, Fig. 6 presents the number of verified sets with respect to the domain size. The hierarchical bitmap index processes only the correct answers to the user query. The signature index performs worse because it has to process some of the

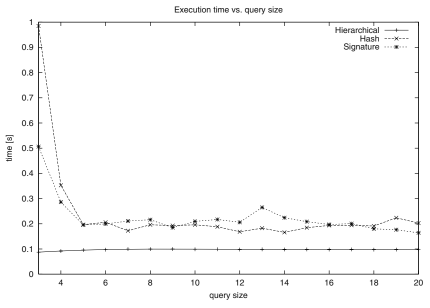


Fig. 3. Search time vs. query size

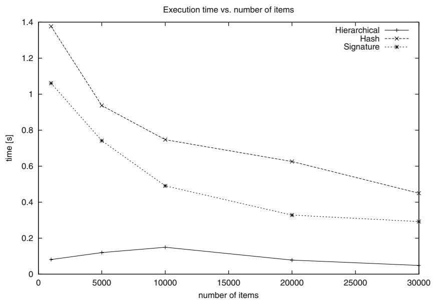


Fig. 4. Search time vs. number of items

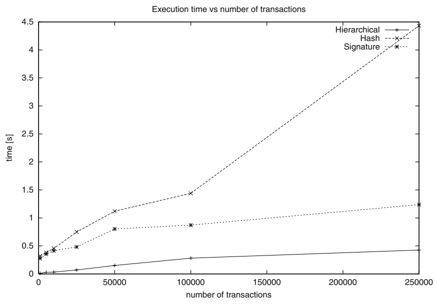


Fig. 5. Search time vs. number of itemsets

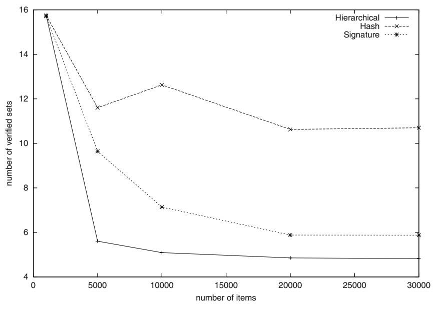


Fig. 6. Number of verified sets vs. number of items

sets that don't actually contain the searched set. Hash index reveals the poorest performance because of the excessive number of false hits (for 30000 different elements almost 70% of sets returned from the index tend to be false hits).

We measured also the performance of the traditional B+ tree index, but we omitted the results because B+ tree index was significantly inferior to all types of set-oriented indexes. For all database sizes and for almost all query sizes queries using B+ tree index ran ten times slower than the queries using the hash index.

6 Other Applications

As we said before the most commonly used class of set-oriented queries are subset queries. Beside subset queries there are two more classes of queries that can be used in terms of set-valued attributes. These are superset and similarity queries.

Superset queries retrieve all sets that are proper subsets of the searched set. Assume that the given set of products is offered at a reduced price. A superset query can be used to find the customers whose market basket is entirely included in the reduced product set, i.e., the customers who visited the shop only to profit from the discount. Another example of the superset query could be: given the pattern describing the correlation of sales of the set of products find all customers who can be characterized using this pattern. Superset queries are also useful in web-based applications. If the searched set consists of web pages and links considered the main navigation path through the web site, the superset query can be used to identify inexperienced users who don't use any advanced features of the web site and who always stay within the main navigation route. Superset search procedure can't use the S-tree structure and has to test all index keys stored in the tree. The algorithm to perform a superset search on all sets contained in the attribute A of the relation R using the searched set q is given below.

```
1: for all $A_i \in R.A$ do
2:    for all levels $l$ do
3:       for all index key nodes $i$ at level $l$ in $A_i$ do
4:          $p = \texttt{skip}(q, l, i)$
5:          if $N_i^l(A_i) \& N_{p+1}^l(q) \neq N_i^l(A_i)$ then
6:             return(false)
7:          end if
8:       end for
9:    end for
10:   return(true)
11: end for
```

This algorithm is a slight modification of the subset search algorithm presented in Sec. 4. This time we are testing all nodes of the index key of S to see if they are entirely contained in the corresponding nodes of q. Function $skip(q, l, i)$ computes the number of nodes in q that must be skipped to reach the node corresponding to $N_i^l(A_i)$. It works exactly as in the subset search algorithm.

The third class of set-oriented queries contains the similarity queries. A similarity query retrieves all sets that are similar to the searched set with the similarity threshold provided by the user. There are many different measures of similarity between two sets. The most commonly used notion of similarity is the Jaccard coefficient which defines the similarity between sets A and B as

$$similarity(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Similarity queries have numerous applications [9]. They can be used to cluster customers into distinct segments based on their buying behavior, to tune direct mail offerings to reach only the desired customers, to make automated recommendations to customers, to dynamically generate web page contents and direct customer-oriented advertisement, etc. Similarity analysis can operate on different levels of similarity. For example, to classify a new customer to one of the existing profiles we are interested in finding the most similar profile. On the other hand, when searching for customers who should be targeted with a new product catalog we are interested in finding those customers, whose purchases are similar to the set of offered products only to some degree (the most similar customers already possess the majority of catalog products, hence they are not the right target for such offer). Finally, in many applications the contrary of the similarity, namely the dissimilarity, can be interesting to analyze. The analysis of the dissimilarity can reveal features and patterns responsible for differences in buying behaviour, web navigation habits, sales anomalies, etc.

For similarity queries no filtering can be applied before visiting all nodes belonging to a given index key. Some existing approaches [9,15] propose methods for efficient S-tree traversal in case of similarity queries. Incorporating those methods into hierarchical bitmap index is subject to our future work. The main idea of the algorithm is to compare all pairs of corresponding nodes and count the number of positions on which both nodes contain '1's. If the percent of common '1's is higher than the user defined threshold then the answer is positive, else negative. The algorithm to perform a similarity search on all sets contained in the attribute A of the relation R using the minimum similarity threshold *min_similarity* is given below.

```
1: for all $A_i \in R.A$ do
2:   $c = 0$
3:   for all levels $l$ do
4:     for all index key nodes $i$ at level $l$ in $q$ do
5:       $p = \texttt{skip}(A_i, l, i)$
6:       $x = N_i^l(A_i) \& N_{p+1}^l(q)$
7:       $c = c + \texttt{count}(x)$
8:     end for
9:   end for
10:  if $\frac{c}{\texttt{numberofkeysinq}} \geq \texttt{min\_similarity}$ then
11:    return(true)
12:  else
13:    return(false)
14:  end if
15: end for
```

For every set A_i the algorithm iterates over all nodes of q and bitwisely ANDs those nodes with corresponding nodes in the index key of A_i. The function count(x) computes the number of bits set to '1' in x. After the comparison of all node pairs is finished the percentage of common positions is calculated. If this percentage exceeds the user defined threshold of minimum similarity *min_similarity* the algorithm adds the given set to the result.

7 Conclusions

Our experiments have proven that existing indexing techniques are not suitable to efficiently index set-valued attributes. Hence, in this paper we have introduced the hierarchical bitmap index. It is a scalable technique to efficiently index large collections of sets. Our index is not affected either by the size of the domain of the indexed attribute or the size of the indexed set. It scales well with the number of sets in the database and exhibits constant response time regardless the query size. Its compactness guarantees that the use of the index is memory efficient. All hierarchical bitmap index features stem from the compact and exact representation of the indexed sets. Because all answers obtained from the hierarchical bitmap index are exact, the hierarchical bitmap index doesn't need any verification steps that are required in case of all other indexing techniques.

This paper presents the results of the initial work on the hierarchical bitmap index. Further improvements will include:

– more sophisticated methods for mapping elements to signature bit positions. Storing elements that frequently occur together in the same index key leaves should lead to the improvement of the index compactness and should increase index filtering capabilities
– applying advanced methods of the S-tree construction to increase pruning at the upper levels of the tree and to improve the selectivity of the tree
– using advanced methods of the S-tree traversal to answer superset and similarity queries
– analyzing possible applications of the hierarchical bitmap index in advanced database querying, e.g., in analytical processing or data mining queries
– developing algorithms for index maintenance

References

1. R. Agrawal, M. J. Carey, C. Faloutsos, S. P. Ghosh, M. A. W. Houtsma, T. Imielinski, B. R. Iyer, A. Mahboob, H. Miranda, R. Srikant, and A. N. Swami. Quest: A project on database mining. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, page 514, Minneapolis, Minnesota, may 1994. ACM Press.
2. M. D. Araujo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing*, pages 2–20, Valparaiso, Chile, 1997. Carleton University Press.
3. C. Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In L. M. Haas and A. Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 355–366, Seattle, Washington, jun 1998. ACM Press.
4. S. Christodoulakis and C. Faloutsos. Signature files: an access method for documents and its analytical performance evaluation. *ACM Transactions on Office Information Systems*, 2(4):267–288, 1984.
5. D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.
6. U. Deppisch. S-tree: a dynamic balanced signature index for office retrieval. In *Proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 77–87, Pisa, Italy, 1986. ACM.
7. C. Faloutsos. Signature files. In *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, 1992.

8. M. Freeston, S. Geffner, and M. H rhammer. More bang for your buck: A performance comparison of bang and r* spatial indexing. In *Proceedings of the Tenth International Conference on Database and Expert Systems Applications DEXA '99*, volume 1677 of *Lecture Notes in Computer Science*, pages 1052–1065. Springer, 1999.
9. A. Gionis, D. Gunopulos, and N. Koudas. Efficient and tunable similar set retrieval. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. ACM Press, jun 2001.
10. A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting*, pages 47–57, Boston, Massachusetts, jun 1984. ACM Press.
11. J. M. Hellerstein and A. Pfeffer. The rd-tree: An index structure for sets. Technical Report 1252, University of Wisconsin at Madison, 1994.
12. S. Helmer and G. Moerkotte. A study of four index structures for set-valued attributes of low cardinality. Technical Report 2/99, Universität Mannheim, 1999.
13. Y. Ishikawa, H. Kitagawa, and N. Ohbo. Evaluation of signature files as set access facilities in oodbs. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 247–256, Washington, D.C., may 1993. ACM Press.
14. T. Morzy and M. Zakrzewicz. Group bitmap index: A structure for association rules retrieval. In R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 284–288, New York, USA, aug 1998. ACM Press.
15. A. Nanopoulos and Y. Manolopoulos. Efficient similarity search for market basket data. *VLDB Journal*, 11(2):138–152, 2002.
16. K. Nørvåg. Efficient use of signatures in object-oriented database systems. In J. Eder, I. Rozman, and T. Welzer, editors, *Proceedings of the 3rd East European Conference on Advances in Databases and Information Systems (ADBIS)*, volume 1691 of *Lecture Notes in Computer Science*, pages 367–381, Maribor, Slovenia, sep 1999. Springer.
17. E. Tousidou, A. Nanopoulos, and Y. Manolopoulos. Improved methods for signature-tree construction. *The Computer Journal*, 43(4):301–314, 2000.

Efficient Monitoring of Patterns in Data Mining Environments

Steffan Baron[1], Myra Spiliopoulou[2], and Oliver Günther[1]

[1] Institute of Information Systems, Humboldt-University Berlin
Spandauer Str. 1, Berlin 10178, Germany
{sbaron,guenther}@wiwi.hu-berlin.de
[2] Institute of Technical and Business Information Systems,
Otto-von-Guericke-Universität Magdeburg
PO Box 4120, Magdeburg 39016, Germany
myra@iti.cs.uni-magdeburg.de

Abstract. In this article, we introduce a general framework for monitoring patterns and detecting interesting changes without continuously mining the data. Using our approach, the effort spent on data mining can be drastically reduced while the knowledge extracted from the data is kept up to date. Our methodology is based on a temporal representation for patterns, in which both the content and the statistics of a pattern are modeled. We divide the KDD process into two phases. In the first phase, data from the first period is mined and interesting rules and patterns are identified. In the second phase, using the data from subsequent periods, statistics of these rules are extracted in order to decide whether or not they still hold. We applied this technique in a case study on mining mail log data. Our results show that a minimal set of patterns reflecting the invariant properties of the dataset can be identified, and that interesting changes to the population can be recognized indirectly by monitoring a subset of the patterns found in the first phase.

1 Introduction and Related Work

With the shift from off-line to on-line business processes and the ongoing development of the Internet as a platform for business transactions, the number and the size of data sets collected electronically has grown tremendously. As a consequence, the investigation in efficient and scalable algorithms for data mining and knowledge discovery in databases (KDD) has become a major research challenge. In recent years, due to the dynamic nature of most data sets, methods and techniques for maintaining and updating previously discovered knowledge have also emerged. One part of this work focuses on the updation of patterns, such as association rules [8,1,14,15], frequent sequences [16], and clusters [9]. They aim at efficiently updating the content of discovered rules, thus avoiding a complete re-run of the mining algorithm on the whole updated data set. Another avenue of research concentrates on the similarity of rules and on the statistical properties of rules by considering the *lifetime* of patterns, i.e., the time in which they are sufficiently supported by the data [10,6,7,11]. For an in depth discussion of algorithms to update mining results and a common framework for them see [3].

However, all of these proposals consider only part of a pattern, either its content, i.e., the relationship in the data the pattern reflects, or the statistical properties of the pattern. In [2], we made a first step towards an *integrated* treatment of these two aspects of a rule. We proposed the *Generic Rule Model* (GRM) which models both the content and the statistics of a rule as a temporal object. Based on these two components of a rule, different types of pattern evolution were defined. Additionally, a simple monitor was implemented which used a user supplied deviation threshold to identify interesting changes to pattern statistics. As shown in this study, the model is general enough to subsume different types of patterns, such as associations, sequences, and clusters. Using this model, an analyst is able to monitor the changes a rule may undergo over time. However, regarding the frequency of applying the mining algorithms to the data, this approach is comparable to the studies mentioned above in that the data is analyzed at regular time intervals. An arbitrary time window is chosen, or data is collected until an arbitrary number of updates are performed, and the mining algorithm is re-applied, potentially limiting the analysis to the data affected by the update. The advantage of this approach is that new patterns are observed as soon as they emerge, and old patterns are removed from the rule base as soon as they become extinct. However, clearly there is a computational overhead when discovering *all* patterns from the data within each time window. We therefore propose the splitting of the KDD process into two phases.[1] In the first phase, a mining algorithm is applied to discover an initial set of patterns. In subsequent periods, a subset of the rules discovered in the first phase is monitored by extracting their statistics using conventional database techniques. Then, changes to the dataset are measured by their impact on the rules being monitored. In some respect, the current work is a follow up of [2] because we also use the GRM to represent patterns. However, the main difference is that in the case study conducted in [2], the mining software was applied regularly to identify emerging or disappearing patterns, and to extract the statistics of the patterns discovered to that point. In the current paper, we use the monitor not only to identify rule changes of a particular strength, but also to check whether old rules still hold. Moreover, instead of mining the data regularly we use changes to the rules being monitored as an indicator of interesting changes.

Usually, not only the patterns themselves are interesting but also the *changes* to these patterns [6,13]. Therefore, it is necessary to update discovered knowledge in the presence of modifications to the underlying data sets. However, due to the huge number of rules discovered this is not always a simple task [12]. Also, the identification of *interesting rules* is widely discussed [7,5]. Our methodology

[1] According to [4] we consider the KDD process as a continuous cycle consisting of (a) identifying the problem to be solved, (b) collecting and analyzing data, (c) interpretation and transposition of the results, and (d) examination of the effectiveness of the actions taken.

reduces the mining effort significantly by limiting the size of the dataset to be analyzed to a minimum. Of course, a domain expert has to identify the optimal time window in advance. But in subsequent periods only the statistics of *interesting* rules have to be examined to decide whether they still hold. Our approach is based on the assumption that strong changes to the whole population affect all patterns in the data. However, instead of monitoring all patterns only the statistics of patterns considered interesting are inspected. Following this approach, we implemented a pattern monitor entirely based on conventional database technology to watch the development of interesting rules over time. The monitor recognizes changes to the whole population indirectly by measuring their impact on the monitored rules. The main advantage results from re-using existing rules by which the indirected discovery of frequent patterns is replaced by directed queries, for example using SQL or a mining language, to extract just their statistics.

The rest of the article is organized as follows. In the next section, our approach for monitoring discovered knowledge is introduced. Section 3 presents a case study on analyzing mail log data in which the usability of the proposed methodology is examined. Section 4 provides a summary and gives a brief outlook on future work.

2 A Methodology for Monitoring Mining Results

In this section we describe our methodology for monitoring the results of the KDD process. It is based on a temporal representation of patterns which consists of a rules' content, i.e., the relationship in the data that the pattern reflects, and the statistical measurements that are captured for the rule.

2.1 Temporal Representation of Patterns

We use the *Generic Rule Model* (GRM) as a basis for the temporal representation of patterns [2]. According to the GRM, a rule R is a temporal object with the following signature:

$$R = (ID, query, timestamp, stats, body, head)$$

In this model, ID is a rule's identifier which is produced by a function that takes the pattern or part thereof as input. With respect to its input, this function generates a unique identifier which can be used to identify the pattern non-ambiguously, even across different mining sessions. Consequently, if for example the body of a rule is used as input, all rules with the same body are referenced by a single identifier. This approach can be used to establish rule groups to be monitored. In *query* the mining query is stored which produced the pattern at *timestamp*. In the simplest case, the mining query consists of a set of constraints which limit the number of discovered rules, e.g. a lower boundary for support and confidence and an upper boundary for the length of the rule. Storing the mining query is essential because results are only comparable as long as the query does not change. If a rule does not fulfill the requirements in the mining query anymore it disappears from the rule base. In *stats*, the statistics of the pattern as a whole are stored, and *body* and *head* denote the antecedent and the consequent of the pattern, respectively. For example, an association rule "$A \Rightarrow B$" with support 25% and confidence 90%, discovered at timestamp t_1 by mining query q is represented by $R_{AB} = (ID_{AB}, q, t_1, [s = 25\%, c = 90\%], A, B)$. Note that in *stats* only the statistics related to the entire rule are stored. Statistics that are captured only for part of the rule, like *body* or *head*, are stored directly in the respective item.

The most important properties of the model are that it (a) represents both the content and the statistics of a rule along with its temporal delimitation, and that it (b) covers results based on different mining paradigms, such as associations, frequent sequences, and clusters [2].

2.2 Pattern Monitoring

The main focus of our work is the monitoring of patterns and the detection of interesting changes without regularly mining the data. We seek to limit the computational effort to a minimum while maintaining the extracted knowledge.

In order to achieve this, we separate the KDD process into two phases, the mining phase which consists solely of the first period, and the monitoring phase which consists of all subsequent periods. In the first phase, after cleaning the data, a mining algorithm is applied to discover rules according to user given thresholds for e.g. confidence and support. From these rules, a rule base is created by importing them into the monitor in which rules are stored according to the GRM. The second phase starts with the selection process in which rules to be monitored are identified. For these rules, statistics are extracted and compared to predefined thresholds. If the statistics of a rule violate the user given thresholds, it is removed from the rule base.

The following schema summarizes the procedure used in the second phase:

1. Select rules to be monitored.
2. Establish rule groups.
3. Specify statistics to be observed for each rule and group.
4. Specify alert thresholds.
5. Regular monitoring of selected rules and/or groups.

1. Select rules to be monitored. In this step, the rules which should be monitored are identified. Usually, this step is highly application dependent. However, in most cases the number of discovered rules will be rather large, and the user needs some guidance to accomplish this task. Choosing the rules to be monitored manually seems to be too time consuming, though it may fit best the needs of most applications. Of course, the user could opt to monitor *all* rules continuously. But this would imply a computational effort comparable to mine the data regularly. As mentioned in Sect. 1, an important feature of a rule is its *interestingness* which is usually a rather *subjective* term. However, if we denote a measure's ability to describe the invariant properties of a dataset as interestingness, it becomes far more *objective*. Therefore, we propose to choose those rules which represent the structure of the dataset, i.e., those rules that are permanently supported by the data. Of course, in the first period it is impossible to say which rules will be permanent. For this reason, all rules produced in the first mining run are potential candidates for the monitoring process. As mentioned above, in realistic applications the number of rules may be huge. However, as the results of our case study show, the number of permanent rules decreases relatively fast with the number of periods. Of course, it may be possible that there are no more permanent rules at one point in time. Therefore, we extend this approach as follows. Depending on the number of initially discovered rules, patterns that disappear from the rule base in subsequent periods could be kept *in memory* until their frequency violates a given threshold, or they are absent for more than a given number of periods. For example, a rule would be removed from memory if it appears in less than 50% of the periods, or if it is absent for more than three periods. Depending on the number of occurrences different rule groups could be defined (e.g., 100%, 90%, 75%, and 50% rules). For example, using these groups, a rule change could be considered interesting if it leads to a group change.

2. Establish rule groups. Many rules are clearly related by content, for example if they have the same antecedent. Therefore, as described in Sect. 2.1, rules can be grouped by choosing an appropriate input for the function that generates the identifier. This step is optional, but it often limits the number of rules to be monitored. For these groups, statistics are captured that potentially give (a) additional insights when interpreting the mining results, and (b) an additional indicator of changes in the rule base. As mentioned in step 1, we used the share of occurrences relative to the total number of periods as grouping criterion.

3. Specify statistics to be observed. In this step the statistical measurements to be captured for single rules as well as for rule groups are selected. Besides the conventional measures *support* and *confidence*, we captured the expected confidence and the lift for single rules.[2] For rule groups, one can also capture the number of rules in a group or aggregated statistics derived from single rule statistics.

4. Specify alert thresholds. In this step threshold values for the measures identified in step 3 are specified. Again, this step depends on the application. Usually, the user will just specify the values that were used to discover the rules in the mining phase. However, a more sophisticated approach is also possible. For ex-

[2] In the context of association rule mining the expected confidence is the ratio of transactions supporting the consequent, i.e., the right hand side of a rule, and the total number of transactions. The lift (*a.k.a.* improvement) of a pattern is defined as the ratio of confidence and expected confidence and gives insights into the quality of the discovered model.

ample, the user could specify a maximum support value for rules that indicate an error, or a deviation value to observe a particular steepness of change.

5. Regular monitoring of selected rules or groups. This step involves the computation of the statistics defined in step 3. It can either be done using a mining language which supports the specification of the patterns to be discovered, or using conventional database technology. In this step the major performance gain is achieved. Instead of mining the data regularly, just the statistics of the monitored rules are collected. Once they are available, derived measures and aggregates are computed and compared to the threshold values specified in step 4. Compared to re-applying the mining algorithm, using a relational DBMS this operation is quite efficient.

While the proposed approach reduces the effort to maintain discovered knowledge it fails to detect new patterns directly since only the statistics of *known* rules are computed. However, this does not seem to be necessary: changes to the distribution of the population can be recognized indirectly by their impact on the statistics of the rules being monitored. Therefore, strong changes to the rule base, either to the number of permanent rules or to the statistics of monitored rules, act as an indicator for significant changes to the population. In such cases, the mining algorithm has to be re-applied to check if new rules have emerged. Only under very rare circumstances, e.g., if the rules being monitored have very small support values, this approach may fail, though it is unlikely that a *permanent* rule has only small support. However, even in that case the problem could be easily bypassed by monitoring the rule groups mentioned in step 1, or by manually selecting a better supported rule.

3 Case Study

In order to show the suitability and efficiency of the proposed methodology we conducted a case study on association rule discovery. For this purpose, we used mail log data as produced by the sendmail mail transport agent which were taken from our mail server.

3.1 Data

For each message sent, sendmail logs one line for incoming mail (fromline), and, depending on the number of recipients, one or more lines for outgoing mail (toline). Amongst other attributes, date and time, addresses of sender and recipient, and the number of recipients are kept. The entire log comprised about 280,000 entries of more than 100,000 messages that passed the mail server within one year. After a first cleaning step the log was anonymized and imported into a relational DBMS, and the attributes to be used for the analysis were chosen.

3.2 Encoding Scheme

In order to ease the analysis, the information given on one log line was combined into a single *multi attribute*. Table 1 shows the encoding scheme used. With respect to daytime, weekday and holidays, the first digit reflects whether a message was sent during regular working hours (`1`) or not (`0`). The following six digits encode the sender of a message (institute, faculty, or extern) and its recipient (institute, faculty, or extern). Clearly, digits 2 to 4 are only set for fromlines, and digits 5 to 7 only for tolines. The last two digits denote the number of recipients which are the same for matching from- and tolines. For example, the multi attribute for a message sent outside working hours where both sender and (one) receiver were internal to the institute is `010000001` for the fromline and `000010001` for the toline. Besides the multi attribute, a unique transaction identifier which was generated automatically while importing the data had been exported to the mining software.

Table 1. Encoding scheme used for the *multi attribute*.

Position	Description
1	working hours (`1`) or not (`0`)
2	senders origin is institute (`1`) or not (`0`)
3	senders origin is faculty (`1`) or not (`0`)
4	senders origin is external (`1`) or not (`0`)
5	receivers origin is institute (`1`) or not (`0`)
6	receivers origin is faculty (`1`) or not (`0`)
7	receivers origin is external (`1`) or not (`0`)
8 9	number of recipients (decimal number)

3.3 Mining Phase

In the mining phase, we used the SAS Enterprise Miner to discover the initial set of rules. We used a time window of one week and imported the transactions into the miner. For the discovery process, we used a minimum confidence of 10% and a minimum support of 5% of the largest single item frequency. For simplicity, we also limited the number of items in a rule to 2. Besides the single item frequencies, we discovered a total of 21 rules which were imported into a relational DBMS. For this purpose, the GRM has been transformed into the relational schema depicted in Fig. 1. In the `samples` table the transaction counts for each sample along with start and end of the corresponding time window are stored. In the `items` table support, lift, and absolute support belonging to that single item with respect to the sample referenced by `sampleid` are stored. In the table `rulestats` statistics with respect to the rules as a whole are stored. Lastly, in the tables `lhs` and `rhs` the items belonging to body and head of the rule referenced by `ruleid` are stored.

```
samples (sampleid, startweek, endweek, transcount)
items (item, sampleid, support, lift, counts)
rulestats (ruleid, expconf, conf, support, lift, counts, sampleid)
lhs (ruleid, item, sampleid)
rhs (ruleid, item, sampleid)
```

Fig. 1. Relational schema for the GRM.

Table 2 shows a sample of rules discovered during the experiments. Basically, there are two different types of rules. The first type shown in the upper part, consists of rules reflecting the *general* mail traffic. These are rules representing messages sent during or outside working hours from senders internal to the institute to one of the three user groups mentioned in Sect. 3.2. For example, the first rule says that if a message is sent during working hours by a member of the institute to two recipients then there is at least one recipient internal to the institute. Rules 2 to 5 express the same relationship for messages sent to one internal or external recipient inside or outside working hours. Compared to rules 2 to 5 the first rule holds with high confidence. Interestingly, there is no significant mail traffic between the institute and the faculty.

The second type of rules as shown in the lower part of Table 2 reflects the *permitted* operations of the mail server, i.e., so-called *alert rules* (cf. Sect.2.2). These rules are expected to exhibit high confidence values. For example, rule 6 says that if a message is sent from outside the faculty to two recipients then at least one of the recipients has to be a member of the institute. Obviously, this rule should always hold with a confidence of 100%. Otherwise, this would indicate that the mail server is being abused e.g. as a spam relay. The remaining alert rules hold with comparably high confidence but not with 100% as expected. However, further investigation of this issue revealed that this was due to internal users who (a) forward their mail to another address using a `.forward` file, or (b) save a local copy of their sent mail through a *blind carbon copy* instead of a *file carbon copy*.

Table 2. Sample of rules in the first period.

No.	LHS	RHS	CONF	SUPPORT	LIFT	COUNTS
1.	110000002	100010002	90.48	2.56	20.05	38
2.	010000001	000010001	68.00	11.45	1.70	170
3.	110000001	100010001	48.88	7.34	1.61	109
4.	110000001	100000101	48.88	7.34	5.63	109
5.	010000001	000000101	31.60	5.32	4.65	79
6.	000100002	000010002	100.00	2.63	25.17	39
7.	101000001	100010001	97.30	2.42	3.20	36
8.	000100001	000010001	94.91	27.61	2.37	410
9.	100100001	100010001	93.87	20.61	3.09	306
10.	000000102	000010002	92.31	2.42	23.23	36

3.4 Monitoring Phase

In the monitoring phase, we firstly identified the rules to be monitored. As described in Sect. 2.2, all the rules discovered from the data of the first time window were chosen. Instead of re-applying the mining algorithm, the statistics for these rules were extracted using conventional database technology. Fig. 2 shows the changes to the number of permanent rules throughout the year. While the number of rules in the second period did not change, in the third period we noticed a slight decrease in the number of rules (19). It can be seen that the number of rules strongly decreases from week 4 to 5 and from week 20 to 21. From week 30 to 31 again two rules disappeared from the set of permanent rules. From the remaining four rules another two rules could be removed since they symmetrically reflected the same relationship in the data. Only the rules with maximum confidence were kept.

At the end of the analysis, the invariant properties of the system are described by just two rules which sometimes covered about 60% of the entire population (Fig. 3). However, while there are two peaks in the number of permanent rules (weeks 5 an 21) which can also be seen in Fig. 3, there is another peak in the support in week 52 which does not have a corresponding peak in the number of permanent rules. In order to check whether this result was caused by a generally decreasing number of rules, we applied the miner to the data of all periods. As Fig. 4 implies, the increase in the support of permanent rules reflects the decrease in the total number of rules. For both weeks 21 and 52, the number of rules decreases as the support of the two rules increases. Apparently, the decrease in the number of rules seems to affect only non-permanent rules. In week 5, we encounter a different behavior, both the number of permanent rules and their support decrease. However, as Fig. 2 and 4 show, the number of permanent rules was equal to the total number of rules in week 5, and the change was caused by

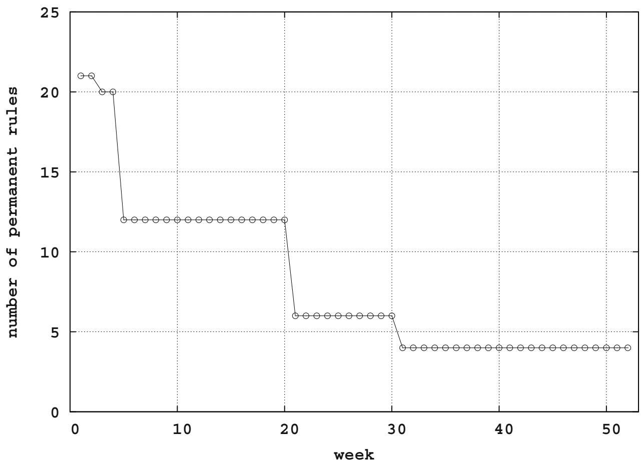


Fig. 2. Number of permanent rules.

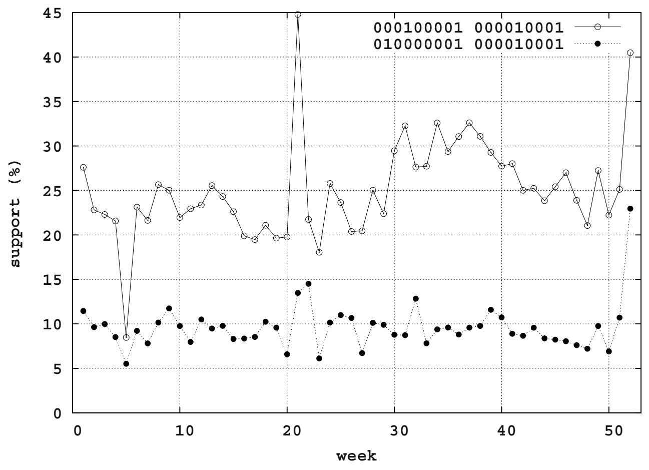


Fig. 3. Support of permanent rules.

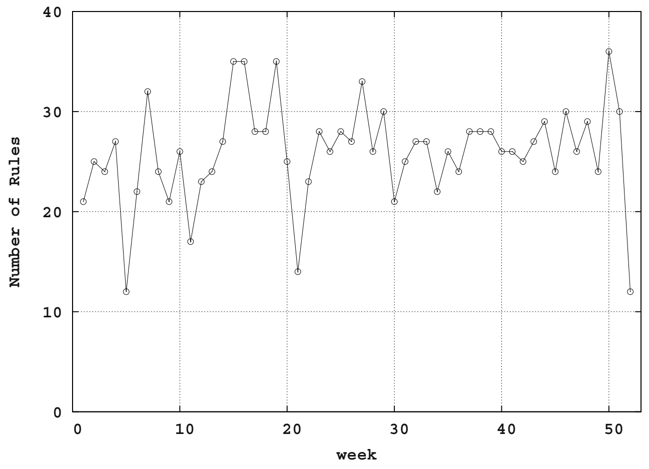


Fig. 4. Total number of rules.

a strong increase in the support of a rule which was permanently supported by the data up to week 5 but not throughout the entire year.

Compared to the total number of rules which constantly amounted to more than 20 in the last 30 periods, the number of permanent rules was very small, which seems to signal that a more sophisticated approach is appropriate. One possible method to cover this issue is to model the stability of rules using certain levels of occurrence as described in Sect. 2.2. However, due to space limitations this is not included here.

In summary, using the proposed approach, it is possible to identify interesting changes within a rule base without mining the data at regular time intervals. Instead, conventional database technology can be used to compute only the statistics of a set of preselected rules. Compared to re-applying the mining algorithm in each time window, this methodology significantly reduces the computational effort required.

3.5 Performance Issues

Obviously, in terms of correctness the results are as good as applying the miner at regular time intervals as long as the same thresholds are used in both the mining phase and the monitoring phase. With respect to performance, we need to consider two different aspects. Firstly, the main factor influencing the performance of the proposed approach is the number of rules being monitored. As already pointed out in Sect. 2.2, the monitor has to compute only the statistics of the rules which have been discovered in the first phase. Changes to the population which may result in new patterns are recognized indirectly by changes to the patterns being monitored. Thus, for each pattern under consideration three SQL queries have to be issued, one that determines the total transaction count for the given period, one that computes the number of transactions which contain both the *lhs* and *rhs* of the rule, and one that computes the number of transactions which contain the *lhs* of the rule. Since the total transaction count for a given period has to be computed only once, the complexity of the approach depends linearly on the number of rules n being monitored, i.e., $O(2n + 1)$. On the other hand, an association rule mining algorithm, e.g. A priori has to scan the database several times in each mining run. Thus, the performance of the miner depends on the number of transactions, which is normally much larger than the number of monitored rules. In our experiments, the time required to process a single SQL query was always close to zero. For example, in the third period there were 6,000 transactions in total, and there were 20 rules being monitored. In order to determine the statistics of those rules, we issued $2 \times 20 + 1$ queries each taking between 0 and 5 milliseconds (1 millisecond on average, 0.105 seconds in total). In comparison, the miner took on average 2.0045 seconds to analyze the dataset. Although these figures do not take into account the overhead of generating the SQL queries or the work needed to import the data into the miner nor to postprocess the discovered rules, they outline the achievable performance gains.

The second aspect concerns the case of strong changes to the rule base which signals that a new mining run is due. In such cases, the proposed approach involves a slight overhead. At first, the statistics of the monitored rules are computed. Then, after detecting strong changes, the miner has to be re-applied in order to investigate the changes in the population that led to rule changes. However, throughout the entire analysis there were only three periods (5, 21, and 52) in which a new mining run were necessary.

4 Conclusions and Future Work

In this study, we introduced a methodology for efficiently maintaining data mining results. Based on a temporal representation for patterns in which both the content and the statistics of a pattern are modeled, we divide the KDD process into two separate phases. In the first phase, an appropriate mining algorithm is applied to discover an initial set of rules. In the second phase, a subset of these patterns is monitored in order to notice interesting changes in the population. As opposed to existing works in this research area, we do not mine the data at regular intervals. Instead, we mine only the data from the first time window of the analysis. Data from subsequent periods is analyzed only by means of determining item counts. The major performance gain is achieved by replacing the indirected discovery process of a mining software with the directed extraction of the statistics. This can be done either using a mining language which supports the specification of the patterns to be discovered or using conventional database techniques like SQL.

While the effort spent on data mining is reduced, the effort to generate the SQL statements to compute the statistics grows with the number of patterns being monitored. However, since these statements are generated automatically from the set of selected patterns this step should be less time consuming, even when monitoring all rules. In a case study we have shown that the proposed methodology (a) helps to identify the minimal set of patterns that reflect the properties of the dataset analyzed, and (b) is able to determine changes to dataset by measuring their impact on the rules monitored. However, it turned out that a more sophisticated approach might be useful to identify also emerging patterns. For example, the miner could be applied periodically, or regularly in a training phase. After identifying a number of e.g. 90% rules, these could be used to monitor changes to the dataset.

Besides quantifying the performance gain of the proposed techniques on a larger dataset, other challenging directions for future work include the application of the methodology to streaming data, and the identification of interdependencies between rules from different data partitions where each partition constitutes a transaction in each of which a number of items (rules) appear.

References

1. N. F. Ayan, A. U. Tansel, and E. Arkun. An Efficient Algorithm To Update Large Itemsets With Early Pruning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–291, San Diego, CA, USA, August 1999. ACM.
2. S. Baron and M. Spiliopoulou. Monitoring Change in Mining Results. In *Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery*, Munich, Germany, Sep. 2001. Springer.
3. S. Baron and M. Spiliopoulou. Monitoring the Results of the KDD Process: An Overview of Pattern Evolution. In J. Meij, editor, *Dealing with the data flood: mining data, text and multimedia*, chapter 6. STT Netherlands Study Center for Technology Trends, The Hague, Netherlands, Apr. 2002.

4. M. J. Berry and G. Linoff. *Data Mining Techniques: For Marketing, Sales and Customer Support*. John Wiley & Sons, Inc., 1997.
5. Y. M. Bing Liu and R. Lee. Analyzing the interestingness of association rules from the temporal dimension. In *IEEE International Conference on Data Mining (ICDM-2001)*, pages 377–384, Silicon Valley, USA, November 2001.
6. S. Chakrabarti, S. Sarawagi, and B. Dom. Mining Surprising Patterns Using Temporal Description Length. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98*, pages 606–617, New York City, NY, Aug. 1998. Morgan Kaufmann.
7. X. Chen and I. Petrounias. Mining Temporal Features in Association Rules. In *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, Lecture Notes in Computer Science, pages 295–300, Prague, Czech Republic, September 1999. Springer.
8. D. W. Cheung, S. Lee, and B. Kao. A General Incremental Technique for Maintaining Discovered Association Rules. In *DASFAA'97*, Melbourne, Australia, Apr. 1997.
9. M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 323–333, New York City, New York, USA, August 1998. Morgan Kaufmann.
10. V. Ganti, J. Gehrke, and R. Ramakrishnan. A Framework for Measuring Changes in Data Characteristics. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 126–137, Philadelphia, Pennsylvania, May 1999. ACM Press.
11. V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and Monitoring Evolving Data. In *Proceedings of the 15th International Conference on Data Engineering*, pages 439–448, San Diego, California, USA, February 2000. IEEE Computer Society.
12. S. Jaroszewicz and D. A. Simovici. Pruning redundant association rules using maximum entropy principle. In *Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference, PAKDD'02*, pages 135–147, Taipei, Taiwan, May 2002.
13. B. Liu, W. Hsu, and Y. Ma. Discovering the set of fundamental rule changes. In *7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 335–340, San Francisco, USA, August 2001.
14. E. Omiecinski and A. Savasere. Efficient Mining of Association Rules in Large Databases. In *Proceedings of the British National Conference on Databases*, pages 49–63, 1998.
15. S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 263–266, Newport Beach, California, USA, Aug. 1997.
16. K. Wang. Discovering Patterns from Large and Dynamic Sequential Data. *Intelligent Information Systems*, 9:8–33, 1997.

FCBI: An Efficient User-Friendly Classifier Using Fuzzy Implication Table

Chen Zheng[1] and Li Chen[2]

[1] Department of Computer Science, National University of Singapore
3 Science Drive 2, Singapore 117543
iscp1232@nus.edu.sg

[2] Department of Business Administration, Alfred University
38 West University Street, NY 14802
lichencs@yahoo.com

Abstract. In the past few years, exhaustive search method under the name of association rule mining has been widely used in the field of classification. However, such kind of methods usually produce too many crisp if-then rules and is not an efficient way to represent the knowledge, especially in real-life data mining application. In this paper, we propose a novel associative classification method called FCBI, i.e., Fuzzy Classification Based on Implication. This method partitions the original data set into fuzzy table without discretization on continuous attributes, the rule generation is performed in the relational database system by using fuzzy implication table. The unique features of this method include its high training speed and simplicity in implementation. Experiment results show that the classification rules generated are meaningful and explainable.

1 Introduction

Building an efficient classifier is one of the essential tasks of data mining research. Previous researchers have developed many heuristic/greedy search based techniques. For example, decision trees [10], rule learning [14], naïve-bayes classification [16]. These techniques aim to find a representative subset of rules that exist in a dataset. In the past few years, the exhaustive search method has been used to select from all the rules given a specified frequency and confidence threshold. Such kind of efficient classifiers are CBA [1], CAEP [8], GAC [9], ADT [12]. Although these classification rule mining algorithm are able to use the most accurate rules for classification, the problem exists when they inherit from association mining. The associative classification usually contains a large number of rules and can cause the problem of combinatorial explosion especially when the data set contains many numeric attributes; the traditional solution to the numeric attribute is discretization [5,20]. However, this could cause the “sharp boundary” problem. Over the years, researchers have attempted to overcome the problem of “sharp boundary” through the use of fuzzy sets [2,3,21]. For example, In [3], The fuzzy association mining is used to mine web log to find user access patterns. But the fuzzy method represents an imprecise knowledge. How to discover precise knowledge without being at the risk of combinatorial explosion? In this paper, we propose a new algorithm, FCBI, which is a general model for mining classification rules using the predefined threshold (minimum support and minimum confidence). We make use of the fact that fuzzy sets provide a smooth transition between numeric data and categorical data through the use of Lambda cut. This allows us to avoid the generation of many hard crisp rules.

FCBI is different from the traditional classifier and other fuzzy classifier in that it first transforms the numeric data to the linguistic term by using fuzzy sets. Next, it uses three measures (certainty factor, support and confidence) to represent the knowledge. It is a more general framework and traditional classification is simply a special case of our algorithm. This means our approach is not in conflict with the existing classification method, we will show it later. Compared with the evolutionary algorithms, which are widely used in the fuzzy classification tasks, our algorithm can be easily implemented; the large number of the rules can be stored inside the modern relation database system to achieve scalability. To the best of our knowledge, we haven’t found any similar technique been proposed.

The remainder of this paper is organized as follows. Section 2 describes related work in the classification area. Section 3 introduces the fuzzy lambda-cut and fuzzy certainty rules. Section 4 describes our new algorithm FCBI. Section 5 presents the experiment results. Finally, Section 6 concludes our work.

2 Related Work

The association rule based classification has been widely studied in the past of few years [1,4,12,17]. In [1], the training data is transformed into transactional database and scanned repeatedly. The data are mapped into a set of (attribute, integer) value. [9] use the data aggregation technique to retrieve the rules. However, when the size of data set is large. It is costly to compute the result using the SQL queries with CUBE function. When the number of attributes increases, this method can’t perform to return the complete set of classification rules. The key problem of the above algorithms is that they simply use the threshold to produce a number of if-then rules. In the other hand, our innovative algorithm FCBI can organize the knowledge in an intuitive fuzzy rule format, which can be easily analyzed by our users and reduce the number of rules significantly. Our method is different from the above algorithm in that the original dataset is mapped into a set of (linguistic variable, linguistic term) pairs without discretization before rule generation. All the possible value will be transformed given different levels of certainty, thus a complete classification rule set is generated. Since we use the fuzzy implication table, which can be processed and stored inside the most of the modern relational database system. We integrate the classification with database system to address the scalability problem.

Although we used fuzzy logic techniques to discover the fuzzy classification rules, our method is different from the evolutionary methods such as neuron-nets in the task of fuzzy classification [6,19], genetic algorithm method [13], genetic programming algorithm method [18]. Compared with such complex evolutionary methods, our method can be easily implemented while using the same meaningful linguistic terms to discover the knowledge inside the dataset and reduce the number of rules significantly.

3 Background

3.1 Fuzzy Lambda Cut

Let us consider a fuzzy set A, we can define a lambda-cut set A_λ, where $A_\lambda = \{x \mid \mu_A(x) \geq \lambda\}$. Any element $x \in A_\lambda$ belongs to A with a grade of membership that is greater than or equal to the value λ.

3.2 Fuzzy Classification Rules

We define the fuzzy classification rules as follows:

$$\text{If } (Var_1\ op\ Term_1) \wedge (Var_2\ op\ Term_2) \ldots \wedge (Var_i\ op\ Term_i) \text{ then } class = CTerm_k \tag{1}$$

Here, Var_i is a fuzzy linguistic variable with the same name as an attribute in database, op is the operator, and $Term_i$ is one of the linguistic terms defined for Var_i. $CTerm_k$ is the linguistic term defined for class.

3.3 Certainty Classification Rules

The syntax of the certainty classification rule is defined as follows:

$$\text{“If } X \text{ is } A\text{, then } Y \text{ is } B \text{ with the certainty } \lambda\text{”.} \quad \lambda \in (0,1] \tag{2}$$

Here, A, B is the fuzzy set. Compared with traditional classification rules, our method uses the fuzzy λ-cut concept to generate the certainty rules. For example, if the rule “if age is young, then risk is high” is generated under λ=0.8, this rule is established only in the “0.8-cut” of the original dataset, while those tuples with membership degree less than 0.8 will never be considered.

4 FCBI System

4.1 System Components

The FCBI system consists of the following three main components:
1. The Pre-processing component: First, the membership degree of each attribute is computed (the membership degree of categorical attribute is 1 or 0), then the λ-cut of each continuous attribute is computed and the original data set is transformed into the relation database (replace the numeric attributes with linguistic terms). After transformation, the relation database is passed to the Rule-generation component.
2. The Rule-generation component: It extracts the meaningful rules from the relational database. The support and confidence value of each candidate rule itemset are computed. After this process, rules are pruned according to pre-defined threshold and covered rules. Then the result is sent to the Post-processing component.
3. The Post-processing component: It calculates the strength of each rule itemset and sort the rule itemset. After that, we search the λ-cut, map the fuzzy certainty rules back to the traditional rules with the value of certainty factor, support and confidence.

4.2 Overview of FCBI Algorithm

Association rule mining has been widely studied in the past few years. Two measurements, namely support and confidence, are usually selected for evaluating the quality of association rules. An association rule $P \to Q$ in a transaction database D, where $P \subset I, Q \subset I, P \cap Q = \varnothing$.

$$\text{support}(P \to Q) = |\{T : P \cup Q \subseteq T \,\&\, T \in D\}| / |D| \tag{3}$$

$$\text{confidence}(P \to Q) = |\{T : P \cup Q \subseteq T, T \in D\}| / |\{T : P \subseteq T, T \in D\}| \tag{4}$$

The confidence value can be represented by $|\text{support}(P \cup Q)/\text{support}(P)$ [17].

A logic-oriented approach to model the fuzzy rule "If X is A, then Y is B" uses a generalized implication operator $\mapsto$, i.e. a mapping $[0,1] \times [0,1] \to [0,1]$ which generalizes the classical material implication. Let X and Y be quantitative attributes in the domain of D_X and D_Y respectively. $A(x), B(y)$ denotes membership degree of x, y belongs to fuzzy set A, B respectively. The individual support can be defined as follows [7]:

$$\text{sup}_{[x,\, y]}(A \Rightarrow B) = A(x) \mapsto B(y) \tag{5}$$

The support and confidence of the implication $A \Rightarrow B$ can be defined as follows:

$$\text{sup}(A \Rightarrow B) = \sum_{(x,y)\in D} \min\{A(x), B(y)\} \tag{6}$$

$$conf(A \Rightarrow B) = \sum_{(x,y)\in D} \min\left\{A(x), B(y)\right\} / \sum_{(x,y)\in D} A(x) \tag{7}$$

The objective for classification is to find all rule items that have support value or confidence value larger than the predefined threshold. We assumed that the training data set was stored in a relational database as a table with schema $(A_1, A_2, \ldots A_j, class)$. Two input parameters, minsup (denoted as $\min s$) and minconf (denoted as $\min c$) were given based on the application and the quality of the data. Another parameter λ was used to represent the certainty of the rule. We propose a new algorithm called FCBI to generate fuzzy classification rules based on associations. The outline of the algorithm is that it first initializes λ equal to 1 and the fuzzy rule base to ϕ, then checks for continuous attributes in the database. Sort each continuous attribute if it is not sorted, then the membership degree is generated according to user specification or a given function. For example, the user may specify that age attribute has two linguistic terms: young and old, and if 60 < age < 70, then the membership degree of old is 0.8 and young is 0.1. The membership degree for categorical data is always "1" or "0". Once the membership degree has been specified, the loop is set to perform classification rule mining based on different λ-cuts. The loop will stop after all the λ certainty rules are generated or timeout. After that, the strength of each rule is computed and the algorithm will stop after the fuzzy rules are defuzzified.

Basically, the FCBI algorithm contains the following two steps. (1) We transform the original dataset to fuzzy table according to different levels of certainty in order to give a complete set of mapping of the original data set; (2) we build the classifier and construct the fuzzy implication table from the transformed fuzzy table.

4.2.1 A Complete Set of Transformed Data

Having defined the λ-cut of each attribute, we can transform every possible linguistic terms of the original dataset to a fuzzy table. The possible linguistic term means that if the value of attribute A_j in a tuple d belongs to the λ-cut of linguistic term A_{jk}, we say A_{jk} is a possible linguistic term. Note that there may be several possible linguistic terms of A_j in tuple d. The possible linguistic terms are used to split the original tuple and partition the original large data set to a small fuzzy table. For example, given a tuple (age=40, salary=2000, risk=low) and λ=0.7, 40 belong to λ-cut of young and middle. 2000 belongs to λ-cut of average and high. Young and middle are possible linguistic terms of age, average and high are possible linguistic terms of salary. After combination, we transform the original tuple to four tuples: (young, average, low), (young, high, low), (middle, average, low), (middle, high, low). This step is detailed in Function Split as follows:

```
Procedure Split ($d, A_{jk\lambda}$) ;
Input: $d$ (tuple in training dataset), $A_{jk\lambda}$
begin
    for each $d \in D$ do
      $P \leftarrow \varnothing$ ;
      for every attribute $A_j$ do
          $Arr_j \leftarrow \varnothing$ ;
          for every linguistic term $A_{jk}$ of $A_j$ do
              if value of $A_j \in A_{jk\lambda}$ then
                  insert linguistic term $A_{jk}$ of $A_j$ into $Arr_j$ ;
              else
                  try next linguistic term ;
              endif
          endfor
          if value of $A_j$ not belongs to any $A_{jk\lambda}$ then
              exit for every attribute loop;
          endif; /*exit for loop*/
      endfor;
      for every attribute $A_j$ do
          Choose element $e$ from each array $Arr_j$ ;
          $P \leftarrow P \cup \{e\}$ ;
          get the combination result $P$ and insert into $T$ ;
      endfor;
    endfor;
end.
```

Fig. 1. Function "Split" gets the transformed fuzzy table.

4.2.2 Rule Generation and Pruning

After we transformed the original data set, we begin construct the fuzzy implication table. First of all, let's give the definition of the basic concepts used in this paper.

Definition 1. Fuzzy Implication Table

Fuzzy implication table for a data set D with j linguistic variables $Var_1, Var_2 ... Var_j$ is a table with schema R as follows:

$R\ (Var_1, Var_2 ... Var_j, CTerm_k, \lambda, \sup, conf)$. A row $r_i\ (V_{1i}, V_{2i} ... V_{ji}, CTerm_i, \lambda, \sup_i, conf_i)$ in R represents a fuzzy certainty rule. $V_{mn} (1 \le m \le j)$ is the value of A_m if attribute A_m is categorical or a linguistic term belongs to Var_m if attribute A_m is numeric. $CTerm_i \in \{CTerm_1, CTerm_2, ... CTerm_k\}$ given k classes. $\sup_i, conf_i \in (0,1]$. An example fuzzy certainty rule is as follows:

> **if** (age=young) and (salary=high) **then** creditRisk=low having a support 34%, confidence 90% given a certainty 0.8.

Definition 2. Contradictory Rules
Given two rule itemsets r_1, r_2, $r_1, r_2 \in R$, $r_1\ (V_{11}, V_{21} ... V_{j1}, CTerm_1, \lambda, \sup_1, conf_1)$ and $r_2\ (V_{12}, V_{22} ... V_{j2}, CTerm_2, \lambda, \sup_2, conf_2)$, if the conditional part and λ is the same, while $CTerm_1$ and $CTerm_2$ are complementary. We say the two rules are contradictory rules. We say r_1 override r_2 given the following three situations:

1. r_1 has the higher confidence value.
2. r_1 has the higher support value if r_1, r_2 have same confidence value.
3. r_1 is generated earlier than r_2.

Definition 3. General Rules and Covered Rules
Given two rule itemsets, $r_1 : A \to C$, $r_2 : A' \to C$, $r_1, r_2 \in R$. Here A, A' denotes the conditional part of the rule. A rule $r_1 : A \to C$ is said to be a general rule *w.r.t.* $r_2 : A' \to C$ Given that $A \subseteq A'$. $r_2 : A' \to C$ is a covered rule. If this situation happens, we delete r_2 and keep r_1.

Definition 4. Rank of the Rule
We propose the new measurement called strength, denoted as S to represent the strength of the rule. The strength of a rule r equals to the summation of all the different certainty factors of r. *i.e.* $S = \sum \lambda\ (0 \le \lambda \le 1)$. Given two rule itemsets $r_1, r_2 \in R$. r_1 is said to have a higher rank than r_2 given the following two situations: 1. $S_1 > S_2$; 2. r_1 is generated earlier than r_2 when $S_1 = S_2$.

The main operation of rule generation is to find all rule items that have support or confidence over the predefined threshold. A rule item r is of the form: $A_n\ and\ condset \to y$, $A_n \notin condset$, $condset$ contains $j-1$ attributes at most. Where A_n is the splitting attribute, $condset$ is a set of attributes, y is the class label. The antecedent of r denotes $r.ant$. The support count of antecedent (denoted as $antcount$) of rule r is the sum of the minimum membership degree of the attributes contained in all the tuples in T that have the similar antecedent with r. The support count of consequence (denoted as $conscount$) of rule r is the sum of the minimum membership degree of the attributes contained in all the tuples in T that have the similar antecedent and class label y with r. All of the $A_n.condset$ and y are kept in the fuzzy implication table. The support of rule r is $(conscount/|T|)*100\%$, where $|T|$ is the number of tuples in T. The confidence of rule r is $(conscount/antcount)*100\%$.

```
Function Rule_Gen ($T$);
Input: $T$
Output: $R_\lambda$
Begin
1  for every attribute $A_n$ in $T$ do
2     Choose $A_n$ as the splitting attribute
3     for $\forall P \subseteq condset$, $P \neq \varnothing$ do   /* $P$ contains $K$ attributes*/
4         $r.antcount = 0$, $r.conscount = 0$, $r.conf = 0$;
5         Scan $T$ to search all $r'$;
6         if $(r'.A_n = r.A_n)$ and $(r'.p = r.p)$ then
7             $r.antcount = r.antcount + MIN(m_{nr'}, m_{1r'}, m_{2r'} ... m_{kr'})$;
8         endif;
9      if $(r'.A_n = r.A_n)$ and $(r'.p = r.p)$ and $(r'.y = r.y)$ then
10            $r.conscount = r.conscount + MIN(m_{nr'}, m_{1r'}, m_{2r'} ... m_{kr'})$;
11        endif;  /*end scan*/
12        $r.conf = r.conscount / r.antcount$, $r.\sup = r.conscount / |T|$;
13        Insert each candidate rule $r$ with $r.\sup$, $\lambda$, $r.conf$ to $F$;
14     endfor;
15  endfor;
16  $R_\lambda \leftarrow \{ f \mid f.conscount \geq \min s.|T| \text{ or } f.Conf \geq \min c \}$, $f \in F$;
17    flagcontra=test contradictory rules;
18    if flagcontra =true then
19        Remove contradictory rules in $R_\lambda$;
20    endif;
21  for each rule itemset $f'$, $f$ belongs to $R_\lambda$ do
22     flag cover= check $f'$ is covered by $f$;
23     if flag cover=true then delete $f'$;
24     endif;
25  endfor;
26  Return $R_\lambda$
end.
```

Fig. 2. Function "Rule_Gen" generate, prune the fuzzy classification rules.

Here we use set P to construct the combination result of *condset* part of rule r. For example, if splitting attribute is age, *condset* contains two attributes: salary and education_level, then P can be attribute of salary, education_level or both. The number of P equals to $2^j - 1$. The candidate rule itemset will be inserted into fuzzy implication table. After that, we prune the rule base according to minimum support and minimum confidence, contradictory rules and covered rules. The main computation cost is in the combination of *condset*, which is related to the cardinality of the data set and distinct attribute values. After we generate the fuzzy rule base on λ, we decrease λ to generate another fuzzy rule base, the two fuzzy rule base will merge and the strength of each rule will be calculated. Finally, rules are sorted by the strength and defuzzified.

5 Experiment

To evaluate the accuracy, efficiency and scalability of FCBI, we have tested our system using a set of real life data. In this section, we report our experimental results on FCBI and compare FCBI with two popular classifiers CBA [1] and C4.5 [10]. It shows that FCBI outperforms CBA and C4.5 in terms of average accuracy and scalability in these data sets. All experiments reported are performed on a 500MHz Pentium PC with 128M main memory running on Microsoft Windows2000 and Oracle8i database system.

We experimented the proposed technique on 10 real-life datasets from three different domains. The first dataset is from the medical domain. The dataset contains different kinds of disease. we are given the data including the patients' age, hobbies, inheritance factor, occupation, duration of smoking, personal particulars etc. Our users wanted to know the intuitive fuzzy rules to check with their domain research work. The second set of data is from the bank. The last set is from the education domain. We try to find the interesting pattern from the university students' examination results and other information. We modify the support measurement and adjust it according to distribution of each class in the dataset. Our system successfully identify a particular group of students who spent less time in studying while still outperformed most of the students. In the experiments, All C4.5 parameters are default values. For CBA, we set support and confidence threshold to 1% and 50% respectively and we disable the limit on number of rules. The data are divided into testing (30%) and training samples (70%). Other parameters remain default. For FCBI, the support and confidence thresholds are set the same as CBA, a set of certainty level from 0.4 to 1.0 is defined, difference for each consecutive certainty level is 0.1 and the strength threshold 2.0 is defined. CBA adopts the discretization method in [20]. The accuracy results are shown in Table 1.

Table 1. The accuracy of C4.5, CBA and FCBI

Dataset	# Attr	# class	# Rec	C4.5	CBA	FCBI
Med1	34	2	5699	84.5	86.9	85.8
Med2	23	2	12301	80.8	81.2	84.2
Med3	20	2	16890	83.1	81.7	83.4
Med4	45	3	23208	72.3	73.1	72.9
Bank1	34	4	76052	74.8	76.5	77.2
Bank2	33	4	86065	79.3	83.5	83.8
Bank3	24	3	137413	75.5	76.2	77.3
Bank4	10	3	78564	85.2	87.3	86.1
Edu1	10	3	2510	89.5	93.2	91.3
Edu2	10	3	4673	92.3	94.3	92.1

From the table, out of the 10 data sets, FCBI achieves the best accuracy in 5 datasets and the second best for 4 datasets. In some data sets, e.g. Med2, FCBI wins the second place over 3% in accuracy. From these figures, we can see that FCBI can perform the same level of accuracy as other classifiers.

To test the scalability of FCBI, we compare the elapsed time of CBA and FCBI on 5 data sets. The results are shown in Table 2. In the experiments, the run-time doesn't include preprocessing and I/O time. So the time to construct different level of membership degree in FCBI and the time to discretize and transform the data set in CBA is not included.

Table 2. The elapsed time of CBA and FCBI

Dataset	# Attrs	# class	# Rec	CBA	FCBI
Med1	34	2	5699	57s	81s
Med2	23	2	12301	87s	79s
Med3	20	2	16890	139s	105s
Bank2	33	4	86065	216s	197s
Edu1	10	3	2510	14s	20s
Average				102.6s	96.4s

FCBI also reduce the number of rules significantly because of using fuzzy linguistic terms. Although we provide the defuzzify function to our users, our users prefer to look at the fuzzy rules. One possible reason may be that they are too busy and they can't afford the time and energy to sieve through the thousands of rules generated by CBA. The number of fuzzy rules generated by FCBI and crisp rules generated by CBA is shown in Table 3.

Table 3. The number of rules by CBA and FCBI

Dataset	# attrs	#class	#rec	CBA	FCBI
Med1	34	2	5699	6137	476
Med2	23	2	1230 1	4376	342
Med3	20	2	1689 0	6543	457
Bank2	33	4	8606 5	6230	327
Edu1	10	3	2510	1694	89
Average				4996	338

6 Conclusion

In this paper, we have proposed a new framework, which adopt the existing fuzzy logic techniques to deal with the continuous attribute using fuzzy transformation and can be more general in performing the task of finding classification rules. We explain the design and implementation of the algorithm on top of the relational database system. The system reduce the size of the processing dataset by using fuzzy partition, thus avoid performance degradation during computation. Besides, it can generate more meaningful rules with linguistic terms instead of producing a large number of crisp rules. Our experiments show that FCBI is effective at classification of various kinds of data and has better average classification accuracy and more scalable in discovering the knowledge from large dataset and reduces the number of rules significantly. We are currently further refining our framework and carrying on more experiments.

References

1. B. Liu, et.al. Integrating classification and association rule mining. In Proceedings of the Fourth ACM SIGKDD KDD-98, New York, USA, 80–86,1998.
2. Chan Man Kuok, Ada Wai-Chee Fu, Man Hon Wong: Mining Fuzzy Association Rules in Databases. SIGMOD Record 27(1), 41–46,1998.
3. Cody Wong, Simon Shiu, Sankar Pal, Mining Fuzzy Association Rules for Web access Case Adaptation, Proceedings of the Workshop Program at the Fourth International Conference on Case-Based Reasoning Vancouver, Canada, 2001.
4. D.Meretakis and B. Wüthrich. Extending naïve Bayes classifiers using long itemsets. In Proceedings of 5th ACM SIGKDD KDD-99, California, 165–174, 1999.
5. Dequan Zhou, et.al, Bayesian classifier based on discretized continuous feature space, Proceedings of 4th International Conference on Signal Processing, Vol: 2, 1225–1228, 1998.
6. Didier D. Nauck, Using symbolic data in neuro-fuzzy classification, in Proceedings of NAFIPS 99, (New York), 536–540, 1999.

7. Eyke Hüllermeier, “Implication-Based Fuzzy Association Rules”. PKDD-01, 241-252, 2001
8. G. Dong, X. Zhang, L. Wong, and J. Li. CAEP: Classification by aggregating emerging patterns. In DS'99, (LNCS 1721), Japan, Dec. 1999.
9. H.Lu, and H-Y. Liu. Decision Tables: Scalable Classification Exploring RDBMS Capabilities. VLDB-2000, 373–384, 2000.
10. J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.
11. K. Ali, S. Manganaris, and R. Srikant, Partial classification using association rules. In ACM SIGKDD-97, 115–118, 1997.
12. K. Wang, S. Zhou, and Y. He. Growing decision tree on support-less association rules. In KDD-00, Boston, MA, 265–269, 2000.
13. N. Xiong et.al, Learning Premises of Fuzzy Rules for Knowledge Acquisition in Classification Problems, Knowledge and Information Systems,Vol.4,Issue 1, 96–111, 2002.
14. P. Clark and T. Niblett. The CN2 induction algorithm. Machine Learning, 3:261–283, 1989.
15. Rakesh Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in VLDB-94, Santiago, Chile, 487–499, 1994.
16. Roberto Bayardo, Brute-Force mining of high confidence classification rules. In Proceedings of 3rd ACM SIGKDD KDD-97, 123–126, 1997.
17. R.Duda, and P.Hart. Pattern Classification and Scene Analysis. Wiley, 1973.
18. Roberto R. F. Mendes et.al, Discovering Fuzzy Classification Rules With Genetic Programming and Co-evolution, PKDD-01, 314–325, 2001.
19. S. K. Pal and S. Mitra, “Multilayer perceptron, fuzzy sets and classification,” IEEE Trans. Neural Networks, vol. 3, 683–697, 1992.
20. U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Proc. IJCAI, 800–805, 1989.
21. Wai-Ho Au; Chan, K.C.C, "FARM: a data mining system for discovering fuzzy associa tion rules" Fuzzy Systems Conference Proceedings (FUZZ-IEEE '99), Vol 3, 1217–1222, 1999.

Dynamic Integration of Classifiers in the Space of Principal Components

Alexey Tsymbal[1], Mykola Pechenizkiy[2], Seppo Puuronen[2], and David W. Patterson[3]

[1]Dept. of Computer Science, Trinity College Dublin, Dublin, Ireland
alexey.tsymbal@cs.tcd.ie

[2]Dept. of Computer Science and Information Systems, University of Jyväskylä
Jyväskylä, Finland
{mpechen, sepi}@cs.jyu.fi

[3]Northern Ireland Knowledge Engineering Laboratory, University of Ulster, U.K.
wd.patterson@ulst.ac.uk

Abstract. Recent research has shown the integration of multiple classifiers to be one of the most important directions in machine learning and data mining. It was shown that, for an ensemble to be successful, it should consist of accurate and diverse base classifiers. However, it is also important that the integration procedure in the ensemble should properly utilize the ensemble diversity. In this paper, we present an algorithm for the dynamic integration of classifiers in the space of extracted features (FEDIC). It is based on the technique of dynamic integration, in which local accuracy estimates are calculated for each base classifier of an ensemble, in the neighborhood of a new instance to be processed. Generally, the whole space of original features is used to find the neighborhood of a new instance for local accuracy estimates in dynamic integration. In this paper, we propose to use feature extraction in order to cope with the curse of dimensionality in the dynamic integration of classifiers. We consider classical principal component analysis and two eigenvector-based supervised feature extraction methods that take into account class information. Experimental results show that, on some data sets, the use of FEDIC leads to significantly higher ensemble accuracies than the use of plain dynamic integration in the space of original features. As a rule, FEDIC outperforms plain dynamic integration on data sets, on which both dynamic integration works (it outperforms static integration), and considered feature extraction techniques are able to successfully extract relevant features.

1 Introduction

Knowledge discovery in databases (KDD) is a combination of data warehousing, decision support, and data mining that indicates an innovative approach to information management. KDD is an emerging area that considers the process of finding previously unknown and potentially interesting patterns and relations in large databases [7]. Current electronic data repositories are growing quickly and contain huge amount of data from commercial, scientific, and other domain areas. The capabilities for collecting and storing all kinds of data totally exceed the abilities to analyze, summarize, and extract knowledge from this data. Numerous data mining methods have recently been developed to extract knowledge from these large databases. Selection of the most appropriate data-mining method or a group of the most appropriate methods is usually not straightforward. Often the method selection is done statically for all new instances of the domain area without analyzing each particular new instance. Usually better data mining results can be achieved if the method selection is done dynamically taking into account characteristics of each new instance.

Recent research has proved the benefits of the use of ensembles of base classifiers for classification problems [6]. The challenge of integrating base classifiers is to decide which of them to select or how to combine their classifications to the final classification.

In many real-world applications, numerous features are used in an attempt to ensure accurate classification. If all those features are used to build up classifiers, then they operate in high dimensions, and the learning process becomes computationally and analytically complicated. For instance, many classification techniques are based on Bayes decision theory or on nearest neighbor search, which suffer from the so-called "curse of dimensionality" [4] due to the drastic rise of computational complexity and classification error in high dimensions [9]. Hence, there is a need to reduce the dimensionality of the feature space before classification. According to the adopted strategy dimensionality reduction techniques are divided into feature selection and feature transformation (also called feature discovery). The variants of the last one are feature extraction and feature construction. The key difference between feature selection and feature transformation is that during the first process only a subset of original features is selected while the second approach is based on a generation of completely new features; feature construction implies discovering missing information about the relationships among features by inferring or creating additional features [14]. Feature extraction is a dimensionality reduction technique that extracts a subset of new features from the original set of features by means of some functional mapping keeping as much information in the data as possible [8].

In this paper, we consider the use of feature extraction in order to cope with the curse of dimensionality in the dynamic integration of classifiers. We propose the FEDIC (Feature Extraction for Dynamic Integration of Classifiers) algorithm, which combines the dynamic selection and dynamic voting integration techniques (DS and DV) with the conventional Principal Component Analysis (PCA) and two supervised eigenvector-based approaches (that use the within- and between-class covariance matrices). The first eigenvector-based approach is parametric, and the other one is nonparametric. Both these take class information into account when extracting features in contrast to PCA [8, 10].

Our main hypothesis is that with data sets, where feature extraction improves classification accuracy when employing a single classifier (such as *kNN* or Naïve Bayes), it will also improve classification accuracy when a dynamic integration approach is employed. Conversely, with data sets, where feature extraction decreases (or has no effect) classification accuracy with the use of a single classifier, then feature extraction will also decrease (or will have no effect) classification accuracy when employing a dynamic integration approach.

In the next section the dynamic integration of classifiers is discussed. Section 3 briefly considers PCA-based feature extraction techniques with respect to classification problems. In Section 4 we consider the FEDIC algorithm, which performs the dynamic integration of classifiers in the transformed space. In Section 5 experiments conducted on a number of data sets from the UCI machine learning repository are described, and the results of the FEDIC algorithm are analyzed and compared to the results of both the static and dynamic selection techniques shown in the nontransformed space.

2 Dynamic Integration of Classifiers

Recently the integration of classifiers has been under active research in machine learning, and different approaches have been considered [6]. The integration of an ensemble of classifiers has been shown to yield higher accuracy than the most accurate base classifier alone in different real-world problems. The two main approaches to integration are: first, the *combination approach,* where base classifiers produce their classifications and the final result is composed using those classifications, and second, the *selection approach,* where one of the classifiers is selected and the final result is the result produced by it.

The most popular and simplest method of combining classifiers is voting (also called majority voting and Select All Majority, SAM) [3]. In this simple method, the classification produced by a base classifier is considered as a vote for a particular class value, and the class value with the most votes is selected as the final classification. Weighted voting (WV) [3] and stacked generalization [25] are examples of more sophisticated combining methods.

One very popular but simple selection approach is CVM (Cross-Validation Majority) [12], which estimates the accuracy of each base classifier using cross-validation and selects a classifier with the highest accuracy.

CVM is an example of a *static* selection method that selects one base classifier for the whole data space. More sophisticated combining and selection methods use the estimates of the local accuracy of the base classifiers or meta-level classifiers, which predict the correctness of base classifiers for a new instance [15, 16]. These more sophisticated selection methods are called *dynamic* selection methods.

In [20] a dynamic approach that estimates the local accuracy of each base classifier by analyzing the accuracies of the base classifiers in near-by instances was elaborated. Instead of training a meta-level classifier that will derive the final classification using the classifications of the base classifiers as in stacked generalization, a meta-level classifier that will estimate the local errors of the base classifiers for each new instance and then use these errors to derive the final classification is trained. To predict the errors of base classifiers, the weighted nearest neighbor classification (WNN) is used [1].

The dynamic integration technique contains two main phases [19]. First, at the learning phase, the training set is partitioned into folds. The cross-validation technique is used to estimate the errors of base classifiers on the training set and a meta-level training set is formed. It contains all the attributes of the training instances and the estimates of the errors of base classifiers on those instances. Second, at the application phase, a combining classifier is used to predict the performance of each base classifier for a new instance.

Two different functions implementing the application phase were considered in [19]: dynamic selection (DS) and dynamic voting (DV). At the application phase, DS selects a classifier with the least predicted classification error using the WNN procedure. DV uses the local errors as weights for the base classifiers. Then, weighted voting is used to produce the final classification.

3 Feature Extraction for Classification

Feature extraction for classification is a search among all possible transformations for the best one, which preserves class separability as much as possible in the space with the lowest possible dimensionality [2, 8]. In other words, we are interested in finding a projection **w**:

$$\mathbf{y} = \mathbf{w}^T \mathbf{x}, \tag{1}$$

where **y** is a $p' \times 1$ transformed data point, **w** is a $p \times p'$ transformation matrix, and **x** is a $p \times 1$ original data point.

In [18] it was shown that the conventional PCA [10, 23] transforms the original set of features into a smaller subset of linear combinations that account for most of the variance of the original set. Although it is still probably the most popular feature extraction technique, it has a serious drawback, giving high weights to features with higher variabilities, irrespective of whether they are useful for classification or not. This may give rise to the situation where the chosen principal component corresponds to an attribute with the highest variability but has no discriminating power.

The usual approach to overcome this problem is to use some class separability criterion, e.g. the criteria defined in Fisher linear discriminant analysis, and based on the family of functions of scatter matrices:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}, \tag{2}$$

where $\mathbf{S}_B$ is the between-class covariance matrix that shows the scatter of the expected vectors around the mixture mean, and $\mathbf{S}_W$ is the within-class covariance, that shows the scatter of samples around their respective class expected vectors.

A number of other criteria were proposed in [8]. Both parametric and nonparametric approaches optimize the criterion (2) by the use of the *simultaneous diagonalization algorithm* [8]:

1. Transformation of **X** to **Y**: $\mathbf{Y} = \mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{\Lambda}$ and $\mathbf{\Phi}$ are the eigenvalues and eigenvectors matrices of $\mathbf{S}_W$.
2. Computation of $\mathbf{S}_B$ in the obtained **Y** space.
3. Selection of *m* eigenvectors of $\mathbf{S}_B$, $\psi_1,\dots,\psi_m$, which correspond to the *m* largest eigenvalues.
4. Finally, new feature space $\mathbf{Z} = \mathbf{\Psi}_m^T \mathbf{Y}$, where $\mathbf{\Psi} = [\psi_1,\dots,\psi_m]$, can be obtained.

It should be noticed that there is a fundamental problem with the parametric nature of the covariance matrices. The rank of the $\mathbf{S}_B$ is at most the *number of classes-1*, and hence no more than this number of new features can be obtained.

The nonparametric method overcomes this problem by trying to increase the number of degrees of freedom in the between-class covariance matrix, measuring the between-class covariances on a local basis. The k-nearest neighbor (kNN) technique is used for this purpose.

The algorithm for nonparametric feature extraction is the same as for the parametric extraction. Simultaneous diagonalization is used as well, and the only difference is in calculating the between-class covariance matrix. In the nonparametric case the between-class covariance matrix is calculated as the scatter of the samples around the expected vectors of other classes' instances in the neighborhood. Two parameters (*nNN* and α) are used to assign more weight to those elements of the matrix, which involve instances lying near the class boundaries and thus being more important for classification. In [8] the parameter α was set to 1 and *nNN* to 3, but without any strict justification. In [20] it was shown that these parameters have different optimal values for each data set.

4 Dynamic Integration of Classifiers with Instance Space Transformation

In order to address the curse of dimensionality in the dynamic integration of classifiers, we propose the FEDIC (Feature Extraction for Dynamic Integration of Classifiers) algorithm that first performs feature extraction and then uses a dynamic scheme to integrate classifiers.

4.1 Scheme of the FEDIC Algorithm

In Figure 1, a scheme that illustrates the components of the FEDIC approach is presented. The FEDIC learning model consists of five phases: the training of the base classifiers phase, the feature extraction phase (FE), the dynamic integration phase (DIC), the model validation phase, and the model testing phase. The model is built using a wrapper approach [11] where the variable parameters in FE and DIC can be adjusted to improve performance as measured at the model validation phase in an iterative manner. These parameters include the threshold value that is related to the amount of covered variance by the first principal components and thus defines the number of output features in the transformed space (it is set up for each feature extraction method); the optimal values of α and *nNN* parameters (as described in Section 3) in the nonparametric feature extraction technique and the number of nearest neighbors in DIC as described later.

In the next subsections we consider the training, feature extraction and dynamic integration of classifiers phases of the FEDIC algorithm.

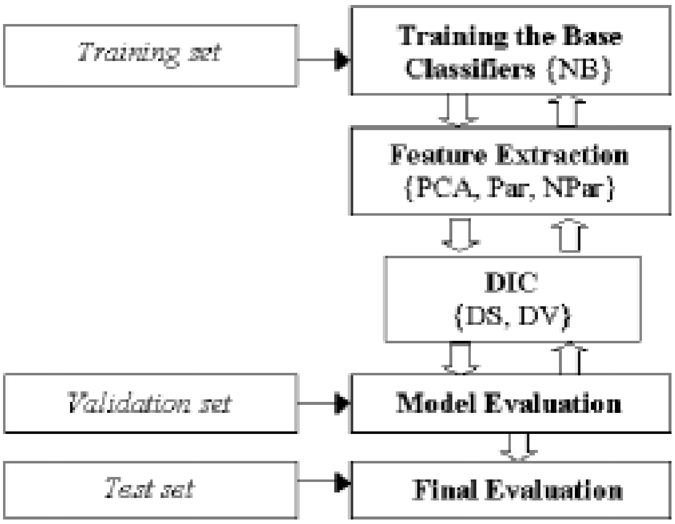


Fig. 1. Scheme of the FEDIC approach

4.2 The Training of the Base Classifiers Phase

The training phase begins with preprocessing which includes categorical features' binarization. Each categorical feature is replaced with a redundant set of binary features, each corresponding to a value of the original feature. An ensemble of classifiers is built from the pre-processed training data as shown in Figure 2. The training set **T** is partitioned into *v* folds. Then, cross-validation is used to estimate the errors of the base classifiers E_j ($\mathbf{x}^*$) on the training set and the meta-level training set $\mathbf{T}^*$ is formed. It contains the attributes of the training instances $\mathbf{x}_i$ and the estimates of the errors of the base classifiers on those instances $E_j(\mathbf{x}^*)$. The learning phase continues with training the base classifiers C_j on the whole training set.

```
T      training set
T_i    i-th fold of the training set
T*     meta-level training set
c(x)   classification of instance x
C      set of base classifiers
C_j    j-th base classifier
C_j(x) prediction of C_j on instance x
E_j(x) estimation of error of C_j on instance x
E      error matrix
m      number of base classifiers
procedure training_phase(T,C)
```

```
begin {fill in the meta-level training set $\mathbf{T}^*$}
  partition $\mathbf{T}$ into $v$ folds
  loop for $\mathbf{T}_i \subset \mathbf{T}$, $i = 1,\ldots,v$
    loop for $j$ from 1 to $m$
      train($C_j$,$\mathbf{T}-\mathbf{T}_i$)
    loop for $\mathbf{x} \in \mathbf{T}_i$
      loop for $j$ from 1 to $m$
        compare $C_j(\mathbf{x})$ with $c(\mathbf{x})$ and derive $E_j(\mathbf{x})$
    collect ($E_1(\mathbf{x})$,...,$E_m(\mathbf{x})$) into $\mathbf{E}$
  $\mathbf{T}^*=\mathbf{T}|\mathbf{E}$
  loop for $j$ from 1 to $m$
    train($C_j$,$\mathbf{T}$)
end
```

Fig. 2. The training phase

4.3 The Feature Extraction Phase

During this phase, feature extraction techniques are applied to the meta-level training set $\mathbf{T}^*$ to produce transformed meta-data with a reduced number of features. The pseudo-code of this process is shown in Figure 3. The formed meta-level data set is the input for the FE module where one of the three functions used: (1) *PCA_FE* that implements conventional PCA, (2) *Par_FE* that implements parametric feature extraction, or (3) *NPar_FE* that implements nonparametric feature extraction. The function *getYspace* is used to calculate an intermediate transformed space needed for the parametric and nonparametric approaches.

$\mathbf{T}^*$, $\mathbf{T}^{*\prime}$	meta-level and transformed meta-level data sets
$\mathbf{S}$, $\mathbf{S}_b$, $\mathbf{S}_w$	total, between- and within-class covariance matrices
$\mathbf{m}$	mean vector
$\mathbf{Y}$	intermediate transformed space
$\mathbf{\Lambda}$, $\mathbf{\Phi}$	eigenvalues and eigenvectors matrices
threshold	the amount of variance in the selected PCs

```
function PCA_FE($\mathbf{T}^*$,threshold) returns $\mathbf{T}^{*\prime}$
begin
    $\mathbf{S} \leftarrow \sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T$
    $\mathbf{\Lambda},\mathbf{\Phi} \leftarrow \mathrm{eigs}(\mathbf{S})$ {the eigensystem decomposition}
    $\mathbf{P_{PCA}} \leftarrow$ formP$_{PCA}$(threshold, $\mathbf{\Lambda}$, $\mathbf{\Phi}$)
        {forms the transformation matrix}
    return $\mathbf{T}^{*\prime} \leftarrow \mathbf{P_{PCA}}\mathbf{T}|\mathbf{E}$
end
```

```
function Par_FE($\mathbf{T}^*$,threshold) returns $\mathbf{T}^{*\prime}$
begin
      $\mathbf{Y} \leftarrow$ getYspace($\mathbf{T}$)
      $\mathbf{S}_B \leftarrow \sum_{i=1}^{c} n_i(\mathbf{m}^{(i)} - \mathbf{m})(\mathbf{m}^{(i)} - \mathbf{m})^T$ {computing of $\mathbf{S}_b$ in the Y
      space}
      $\mathbf{\Lambda},\mathbf{\Phi} \leftarrow \mathrm{eigs}(\mathbf{S}_B)$
      $\mathbf{P}_{Par} \leftarrow$ formP$_{\mathrm{Par}}$(threshold, $\mathbf{\Lambda}$, $\mathbf{\Phi}$)
      return $\mathbf{T}^{*\prime} \leftarrow \mathbf{P}_{Par}\mathbf{Y} \mid \mathbf{E}$
end
function NPar_FE($\mathbf{T}^*$,threshold, kNN, alpha) returns $\mathbf{T}^{*\prime}$
begin
      $\mathbf{Y} \leftarrow$ getYspace($\mathbf{T}^*$)
      $w_{ik} \leftarrow \frac{\min_j\{d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})\}}{\sum_{j=1}^{c} d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})}$
      $\mathbf{S}_B \leftarrow \sum_{i=1}^{c} n_i \sum_{k=1}^{n_i} w_{ik} \sum_{j=1, j \neq i}^{c} (\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})(\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})^T$
      $\mathbf{\Lambda},\mathbf{\Phi} \leftarrow \mathrm{eigs}(\mathbf{S}_B)$
      $\mathbf{P}_{NPar} \leftarrow$ formP$_{\mathrm{NPar}}$(threshold, $\mathbf{\Lambda}$, $\mathbf{\Phi}$)
      return $\mathbf{T}^{*\prime} \leftarrow \mathbf{P}_{NPar}\mathbf{Y} \mid \mathbf{E}$
end
function getYspace($\mathbf{T}^*$) returns Y
begin
      $\mathbf{S}_W \leftarrow \sum_{i=1}^{c} n_i \sum_{j=1}^{n_i} (\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})(\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})^T$
      $\mathbf{\Lambda},\mathbf{\Phi} \leftarrow \mathrm{eigs}(\mathbf{S}_w)$
      return $\mathbf{Y} \leftarrow \mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$
end
```

Fig. 3. The feature extraction phase

4.3 The Dynamic Integration Phase

The transformed meta-data **T***' is the input for the DIC module where the application phase of the dynamic integration process is performed - the combining classifier is used to predict the performance of each base classifier for a new instance. Either the function *DS_application_phase* or the function *DV_application_phase* (Figure 4) is used for this purpose. Both functions begin with finding in the training set **T***' a nearest neighborhood **NN** of the test instance **x** transformed with the corresponding transformation matrix calculated at the FE phase. The first function *DS_application_phase* implements Dynamic Selection. In the DS application phase the classification error E_j^* is predicted for each base classifier C_j using the WNN procedure and a classifier with the lowest error (with the least global error in the case of ties) is selected to make the final classification. The second function *DV_application_phase* implements Dynamic Voting. In the DV application phase each base classifier C_j receives a weight W_j that depends on the local classifier's performance, and the final classification is conducted by voting classifier predictions $C_j(\mathbf{x})$ with their weights W_j. For both functions the size of the set **NN** is an adjustable parameter.

```
x' an instance x transformed with $P_{PCA}$, $P_{NPar}$ or $P_{NPar}$
W  vector of weights for base classifiers
T*' transformed meta-level training set
$E^*_j(\mathbf{x})$ prediction of error of $C_j$ on instance x

function DS_application_phase(T*',C,x) returns class of x
    begin
      NN=FindNeighborhood(T*',x',nn)
      loop for j from 1 to m
        $E_j^* \leftarrow \frac{1}{nn}\sum_{i=1}^{nn} W_{NN_i}\cdot E_j(\mathbf{x}_{NN_i})$ {WNN estimation}
      $l \leftarrow \arg\min_j E_j^*$ {number of cl-er with min. $E_j^*$}
      {with the least global error in the case of ties}
      return $C_l(\mathbf{x})$
    end

function DV_application_phase(T*',C,x) returns class of x
    begin
      NN=FindNeighborhood(T*',x',nn)
      loop for j from 1 to m
        $W_j \leftarrow 1 - \frac{1}{nn}\sum_{i=1}^{nn} W_{NN_i}\cdot E_j(\mathbf{x}_{NN_i})$ {WNN estimation}
      return Weighted_Voting(W,$C_1(\mathbf{x})$,...,$C_m(\mathbf{x})$)
    end
```

Fig. 4. The dynamic integration phase

We do not devote a separate subsection to the model validation and model evaluation phases since they are performed in a straightforward manner. At the validation phase, the performance of the given model with the given parameter settings on an independent validation data set is tested. The given model, its parameter settings and performance are recorded. And at the final evaluation phase, the best model from the number of obtained models is selected. This model is the one with optimal parameters settings as based on the validation results. The selected model is then tested with a test data set.

In the next section we consider our experiments where we analyzed and compared the feature-extraction techniques described above.

5 Experimental Studies

We conducted the experiments on 20 data sets with different characteristics taken from the UCI machine learning repository [5]. The main characteristics of the data sets, which include the name of a data set, the number of instances included in a data set, the number of different classes of instances, and the numbers of different types of features (categorical and numerical) included in the instances were presented in [21]. In [22] results of experiments with feature subset selection techniques with the dynamic selection of classifiers using these data sets were presented.

For each data set 70 test runs were made. In each test run a data set was first split into the training set, the validation set, and the test set by stratified random sampling. Each time 60 percent of the instances were included in the training set. The other 40 percent were divided into two sets of approximately equal size (the validation and test sets). The validation set was used in the iterative refinement of the ensemble. The test set was used for the final estimation of the ensemble accuracy.

To construct ensembles of base classifiers we have used the EFS_SBC (Ensemble Feature Selection for the Simple Bayesian Classification) algorithm, introduced in [20]. Initial base classifiers were built using the Naïve Bayes on the training set and later refined using a hill-climbing cycle on the validation data set. The size of ensemble was selected to be equal to 25. It was shown that the biggest gain is achieved already with this number of base classifiers [2]. The diversity coefficient α was selected as it was recommended in [20] for each data set.

At each run of the algorithm, we collected accuracies for the four types of integration of the base classifiers: Static Selection (SS), Weighted Voting (WV), Dynamic Selection (DS), and Dynamic Voting (DV). In dynamic integration, the number of nearest neighbors for the local accuracy estimates was pre-selected from the set of six values: 1, 3, 7, 15, 31, 63 ($2^n - 1, n = 1,...,6$), for each data set separately. Heterogeneous Euclidean-Overlap Metric (HEOM) [24] was used for calculation of the distances in dynamic integration.

A multiplicative factor of 1 was used for the Laplace correction in simple Bayes. Numeric features were discretized into ten equal-length intervals (or one per observed value, whichever was less). Software for the experiments was implemented using the MLC++ machine learning library [13].

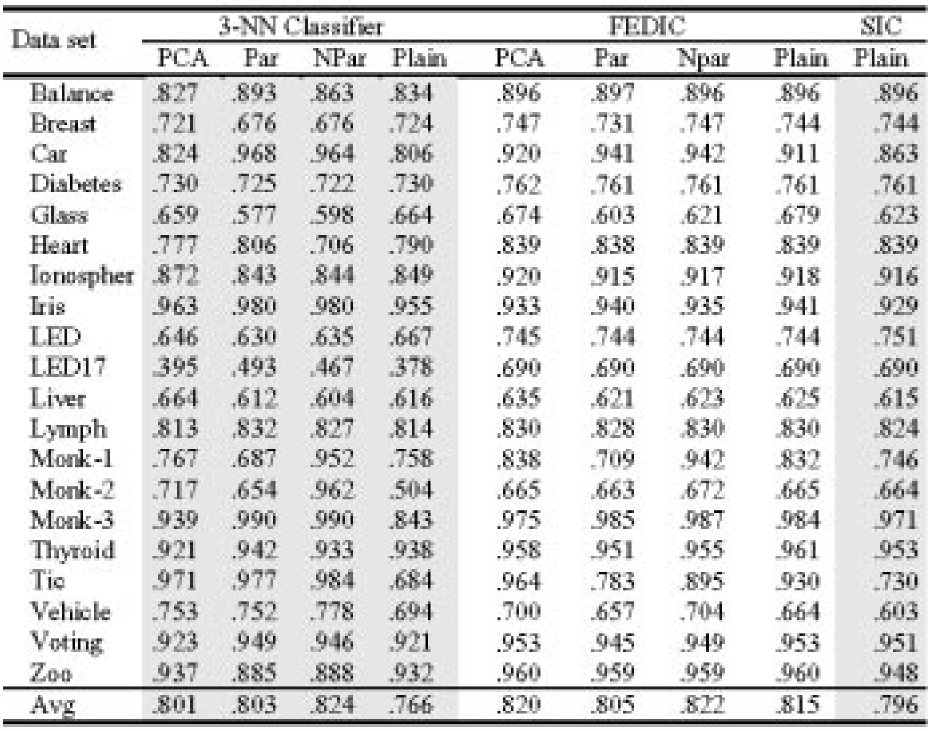
In Section 6 the results of dynamic integration with feature extraction using FEDIC are compared with the results when no feature extraction was used, and dynamic integration was therefore carried out in the space of original features.

6 Results and Discussions

The basic results of the experiments are presented in Table 1. The average classification accuracies of the 3-NN classifier (3-nearest neighbor search) are given for the three feature extraction techniques, namely PCA, parametric (Par) and non-parametric (NPar) approaches. Additionally, classification accuracy for the situation without feature extraction (Plain) is also shown. Then, in the same order, accuracies for the FEDIC approaches averaged over dynamic selection and dynamic voting schemes are presented. The last column contains classification accuracies for the static integration of classifiers (SIC) averaged over static selection and weighted voting. Each row of the table corresponds to a single data set. The last row includes the results averaged over all the data sets.

From Table 1 one can see that the nonparametric approach has the best accuracy on average both with the base classifier and with the dynamic integration scenarios. Although the parametric approach has extracted the least number of features, and it has been the least time-consuming approach, its performance has been unstable. The parametric approach has rather weak results on the Glass, Monk-1, and Tic data sets in comparison to the other feature extraction approaches. The scenario with parametric feature extraction has the worst average accuracy.

Table 1. Results of the experiments



Data set	3-NN Classifier				FEDIC				SIC
	PCA	Par	NPar	Plain	PCA	Par	Npar	Plain	Plain
Balance	.827	.893	.863	.834	.896	.897	.896	.896	.896
Breast	.721	.676	.676	.724	.747	.731	.747	.744	.744
Car	.824	.968	.964	.806	.920	.941	.942	.911	.863
Diabetes	.730	.725	.722	.730	.762	.761	.761	.761	.761
Glass	.659	.577	.598	.664	.674	.603	.621	.679	.623
Heart	.777	.806	.706	.790	.839	.838	.839	.839	.839
Ionospher	.872	.843	.844	.849	.920	.915	.917	.918	.916
Iris	.963	.980	.980	.955	.933	.940	.935	.941	.929
LED	.646	.630	.635	.667	.745	.744	.744	.744	.751
LED17	.395	.493	.467	.378	.690	.690	.690	.690	.690
Liver	.664	.612	.604	.616	.635	.621	.623	.625	.615
Lymph	.813	.832	.827	.814	.830	.828	.830	.830	.824
Monk-1	.767	.687	.952	.758	.838	.709	.942	.832	.746
Monk-2	.717	.654	.962	.504	.665	.663	.672	.665	.664
Monk-3	.939	.990	.990	.843	.975	.985	.987	.984	.971
Thyroid	.921	.942	.933	.938	.958	.951	.955	.961	.953
Tic	.971	.977	.984	.684	.964	.783	.895	.930	.730
Vehicle	.753	.752	.778	.694	.700	.657	.704	.664	.603
Voting	.923	.949	.946	.921	.953	.945	.949	.953	.951
Zoo	.937	.885	.888	.932	.960	.959	.959	.960	.948
Avg	.801	.803	.824	.766	.820	.805	.822	.815	.796

The nonparametric approach extracts more features due to its nonparametric nature, and still it was less time-consuming than the PCA and Plain classification. We should also point out that feature extraction speeds up dynamic integration in the same way as feature extraction speeds up a single classifier. This is as one could expect, since nearest-neighbor search for the prediction of local accuracies is the most time-consuming part of the application phase in dynamic integration, and it uses the same feature space as a single nearest-neighbor classifier does. Moreover, the two nearest-neighbor search processes (in dynamic integration and in a base classifier) are completely identical, and differ only in the number of nearest neighbors used to define the neighborhood.

The results of Table 1 show that, in some cases, dynamic integration in the space of extracted features results in significantly higher accuracies than dynamic integration in the space of original features. This is the situation with the Car, Liver, Monk-1, Monk-2, Tic-Tac-Toe and Vehicle data sets. For these data sets we have pairwise compared each FEDIC technique with the others and with static integration using the paired Student *t*-test with the 0.95 level of significance. Results of the comparison are given in Table 2. Columns 2-6 of the table contain the results of comparing a technique corresponding to the row of a cell with a technique corresponding to the column, using the paired *t*-test. Each cell contains win/tie/loss information according to the *t*-test. For example, PCA has 3 wins against the parametric extraction, 1 draw and 1 loss on 5 data sets.

Table 2. Results of the paired *t*-test (win/tie/loss information) for data sets, on which FEDIC outperforms plain dynamic integration

	PCA	Parametric	Nonparam.	Plain	SIC
PCA		3/1/1	2/0/3	4/1/0	4/1/0
Parametric	1/1/3		0/2/3	2/2/1	3/1/1
Nonparam.	3/0/2	3/2/0		4/1/0	5/0/0
Plain	0/1/4	1/2/2	0/1/4		1/3/1
SIC	0/1/4	1/1/3	0/0/5	1/3/1	

There is an important trend in the results – the FEDIC algorithm outperforms dynamic integration on plain features only on those data sets, on which feature extraction for classification with a single classifier provides better results than the classification on the plain features. If we analyze this correlation further, we will come to the conclusion that feature extraction influences the accuracy of dynamic integration to a similar extent as feature extraction influences the accuracy of base classifiers. This trend supports our expectations about the behavior of the FEDIC algorithm.

The reason for that behavior is that both the meta-level learning process in dynamic integration, and the base learning process in base classifiers use the same feature space. Though, it is necessary to note, that the output values are still different in those learning tasks (these are local classification errors and the classes themselves correspondingly). Thus, the feature space is the same, and the output values to be predicted are different. This justifies that the influence of feature extraction on the accuracy of dynamic integration in comparison with the influence on the accuracy of a single classifier is still different to a certain degree.

In Figure 5 we summarize the accuracy results obtained on those data sets where at least some FEDIC-based technique (with PCA, parametric or nonparametric feature extraction) significantly outperforms the dynamic integration of classifiers on plain feature sets and the averaged results. It can be seen from the histogram that the non-parametric FEDIC shows the best accuracy on average of all the techniques considered. FEDIC and plain dynamic integration on average show almost the same results although we have to point out that this has happened due to the very unstable behavior of the parametric approach. The dynamic approaches significantly outperform the static ones.

7 Conclusion

Feature extraction as a dimensionality reduction technique helps to overcome the problems related to the “curse of dimensionality“ with respect to the dynamic integration of classifiers. The experiments showed that the DIC approaches based on the plain feature sets had worse results in comparison to the results obtained using the FEDIC algorithm. This supports the fact that dimensionality reduction can often enhance a classification model.

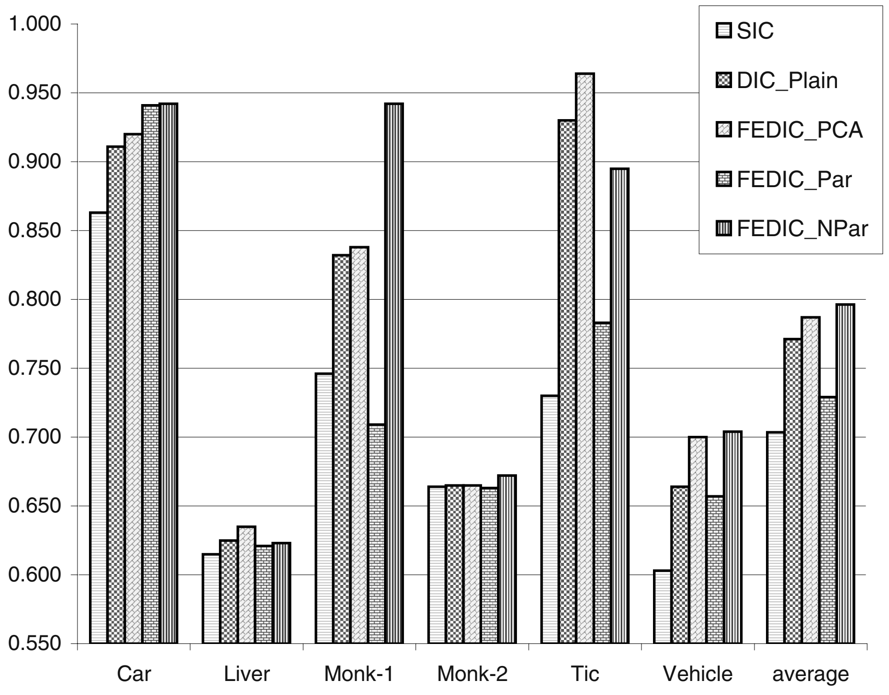


Fig. 5. Classification accuracy for data sets, on which the FEDIC algorithm outperforms plain dynamic integration

The results showed that the proposed FEDIC algorithm outperforms the dynamic schemes on plain features only on those data sets, on which feature extraction for classification with a single classifier provides better results than classification on plain features. When we analyzed this dependency further, we came to a conclusion that feature extraction influenced on the accuracy of dynamic integration in most cases in the same manner as feature extraction influenced on the accuracy of base classifiers.

The nonparametric approach was the best on average; however, it is necessary to note that each feature extraction technique was significantly better than all the other techniques at least on one data set. Further research is needed to define the dependencies between the characteristics of a data set and the type and parameters of the feature extraction approach that best suits it.

Some of the most important issues for future research to be raised by this work, include how the algorithm could automatically determine what transformation matrix should be chosen (i.e. what is the optimal feature extraction method) from the characteristics of the input data and what the optimal parameter settings for the selected feature extraction method should be. Also of interest is, how the most appropriate dynamic integration scheme could be automatically identified.

Acknowledgments. This research is partly supported by Science Foundation Ireland and COMAS Graduate School of the University of Jyväskylä, Finland. We would like to thank the UCI ML repository of databases, domain theories and data generators for the data sets, and the MLC++ library for the source code used in this study.

References

1. Aivazyan, S.A. Applied statistics: classification and dimension reduction. Finance and Statistics, Moscow (1989).
2. Aladjem, M. Parametric and nonparametric linear mappings of multidimensional data. Pattern Recognition, Vol.24(6) (1991), pp. 543–553.
3. Bauer, E., Kohavi, R. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Machine Learning, Vol. 36, Nos. 1,2 (1999) 105–139.
4. Bellman, R., Adaptive Control Processes: A Guided Tour, Princeton University Press (1961).
5. Blake, C.L., Merz, C.J. UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/ MLRepository.html]. Dept. of Information and Computer Science, University of California, Irvine, CA (1998).
6. Dietterich, T.G. Machine learning research: four current directions. AI Magazine 18(4) (1997) 97–136.
7. Fayyad U.M. Data Mining and Knowledge Discovery: Making Sense Out of Data, IEEE Expert, Vol. 11, No. 5, Oct. (1996) pp. 20–25
8. Fukunaga, K. Introduction to statistical pattern recognition. Academic Press, London (1999).
9. Hall, M.A. Correlation-based feature selection of discrete and numeric class machine learning. In Proc. Int. Conf. On Machine Learning (ICML-2000), San Francisco, CA. Morgan Kaufmann, San Francisco, CA (2000) 359–366.
10. Jolliffe, I.T. Principal Component Analysis. Springer, New York, NY. (1986).

11. Kohavi, R. Wrappers for performance enhancement and oblivious decision graphs. Dept. of Computer Science, Stanford University, Stanford, USA. PhD Thesis (1995).
12. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In C.Mellish (ed.), Proc. 14th Int. Joint Conf. on Artificial Intelligence IJCAI-95. Morgan Kaufmann, San Francisco, CA (1995) 1137–1145.
13. Kohavi, R., Sommerfield, D., Dougherty, J. Data mining using MLC++: a machine learning library in C++. In M.G.Radle (ed.) Proc. 8th IEEE Conf. on Tools with Artificial Intelligence. IEEE CS Press, Los Alamitos, CA (1996) 234–245.
14. Liu H. Feature Extraction, Construction and Selection: A Data Mining Perspective, ISBN 0-7923-8196-3, Kluwer Academic Publishers (1998).
15. Merz, C.J. Dynamical selection of learning algorithms. In D.Fisher, H.-J.Lenz (eds.), Learning from data, artificial intelligence and statistics, Springer-Verlag, NY (1996).
16. Merz, C.J. Using correspondence analysis to combine classifiers. Machine Learning 36(1–2) (1999) 33–58.
17. Opitz, D. & Maclin, D. Popular ensemble methods: an empirical study. Journal of Artificial Intelligence Research 11 (1999), 169–198.
18. Oza, N.C., Tumer, K. Dimensionality reduction through classifier ensembles. Technical report NASA-ARC-IC-1999-124, Computational Sciences Division, NASA Ames Research Center, Moffett Field, CA (1999).
19. Puuronen, S., Terziyan, V., Tsymbal, A. A dynamic integration algorithm for an ensemble of classifiers. In Z.W. Ras, A. Skowron (eds.), Foundations of Intelligent Systems: ISMIS'99, Lecture Notes in AI, Vol. 1609, Springer-Verlag, Warsaw (1999) 592–600.
20. Tsymbal, A., Puuronen, S., Patterson, D. Ensemble feature selection with the simple Bayesian classification. Information Fusion, Special Issue “Fusion of Multiple Classifiers”, Elsevier Science (2003) (to appear).
21. Tsymbal A., Puuronen S., Pechenizkiy M., Baumgarten M., Patterson D. Eigenvector-based feature extraction for classification. In Proc. 15th Int. FLAIRS Conference on Artificial Intelligence, Pensacola, FL, USA, AAAI Press (2002) 354–358.
22. Tsymbal A., Puuronen S., Skrypnyk I. Ensemble feature selection with dynamic integration of classifiers. In Int. ICSC Congress on Computational Intelligence Methods and Applications CIMA'2001, Bangor, Wales, U.K (2001).
23. William D.R., Goldstein M. Multivariate Analysis. Methods and Applications. ISBN 0-471-08317-8, John Wiley & Sons (1984), 587 p.
24. Wilson, D.R. & Martinez, T.R. Improved heterogeneous distance functions. Journal of Artificial Intelligence Research 6(1) (1997), 1–34
25. Wolpert, D. Stacked Generalization. Neural Networks, Vol. 5 (1992) 241–259.

Formal Semantics of the ODMG 3.0 Object Query Language[⋆]

Alexandre Zamulin

A.P. Ershov Institute of Informatics Systems
Siberian Division of Russian Academy of Sciences
Novosibirsk 630090, Russia
zam@iis.nsk.su
fax: +7 3832 323494

Abstract. Formal semantics of OQL in terms of object algebra, such as quantification, mapping, selection, unnesting or partitioning, developed by the author is defined, and it is shown in multiple examples that OQL queries can be easily expressed by means of this algebra. As a result, an OQL query can be mechanically translated into the corresponding object algebra expression, which can be further optimized and executed.

Keywords: object algebra, object-oriented database, OQL, formal semantics, query language

1 Introduction

An object data model representing an object-oriented data base as a dynamic system with implicit state has been presented in [3]. This model regards the database state as a many-sorted algebra composed of the states of individual objects. Basing on this model, an object algebra for the Object Data Standard ODMG 3.0 [5] was elaborated and formally defined [8]. It was shown in multiple examples that OQL queries could be easily expressed by means of the proposed facilities and it was proclaimed that the algebra could be used for the formal definition of OQL semantics, which remained a subject of further research. The result of this research is presented in this paper organized as follows.

A brief review of the object data model presented in [3] is given in Section 2. An example database schema used for illustration of object algebra operations is presented in Section 3. Object algebra expressions formally defined in [8] are informally explained in Section 4. The general form of the OQL query is presented in Section 5. The formal semantics of different components of OQL in terms of object algebra is given in Section 6. Related work is reviewed and some conclusions are drawn in Section 7.

[⋆] This research is supported in part by Russian Foundation for Basic Research under Grant 01-01-00787.

2 Object Database

The object data model is based on a type system including a number of basic types (the type Boolean among them) and a number of type constructors including record type, set type, and object type constructors [3]. It is extended in [8] by the bag type constructor and can be easily extended to include other collection type constructors, such as lists and arrays. To save space, we consider only set and bag types in this paper and denote by $col(t)$ each of them where it does not matter which one is actually used.

It is assumed that each type is equipped with conventional operations. Thus, the following operations are applicable to all sets and bags: "$\cup$" (union), "$\cap$" (intersection), "$\subset$" (inclusion), "$\in$" (membership), and *count* (the number of elements). Several operations are applicable to sets and bags of numerical values. These are *avg*, *sum*, *max*, and *min*. The bag type also possesses the operation *setof* converting a bag into a set. A record type $\texttt{rec}\ p_1 : t_1, ..., p_n : t_n\ \texttt{end}$ is equipped with a non-strict record construction operation *rec* producing a record on the base of record field values and projecting operations p_i producing field values of a given record.

An implementation of the data types is a *base algebra*. A sort associated with a data type t in the base algebra $\mathtt{A}$ is denoted by $\mathtt{A_t}$, and the value of an expression e in $\mathtt{A}$ is denoted by $\mathtt{e^A}$. For a data type $set(t)$, the sort $\mathtt{A_{set(t)}}$ consists of sets of elements of $\mathtt{A_t}$. Likewise, for a data type $bag(t)$, the sort $\mathtt{A_{bag(t)}}$ consists of multisets of elements of $\mathtt{A_t}$. For a data type $\texttt{rec}\ p_1 : t_1, ..., p_n : t_n\ \texttt{end}$, the sort $\mathtt{A_{rec\ p_1:t_1,...,p_n:t_n\ end}}$ consists of tuples $\langle \mathtt{v_1}, ..., \mathtt{v_n} \rangle$ where $\mathtt{v_i}$ is either an element of $\mathtt{A_{t_i}}$ or a special value $\perp$ not belonging to any sort.

An *object database schema* consists of a number of *class declarations* consisting of *attribute*, *relationship*, *constructor*, and *method* declarations. A *database state* $\mathtt{A}$ is a many-sorted algebra extending the base algebra by sets of *object identifiers*, $\mathtt{A_c}$, associated with each class c and partial functions $\mathtt{a^A_{ct}} : \mathtt{A_c} \to \mathtt{A_t}$ associated with each attribute or relationship a of type t in class c. The set $\mathtt{A_c}$ is called the *extent* of the class c and the function $\mathtt{a^A_{ct}}$ is called the *attribute function.* The state of an object of a class c is represented by the values produced by the attribute functions of this class. The *database* is defined as a set of states and a set of functions associated with constructor and method definitions, which initialize, observe, or update the state.

There are several rules for creating elementary expressions involving attribute and method names. Thus, if $a : t$ is the declaration of an attribute in a class c, and τ is an expression of type c, then $\tau.a$ is an expression of type t called an *attribute access.* The expression is interpreted in a state $\mathtt{A}$ by invocation of the corresponding attribute function.

If $m : r \to t$ is a method declaration in a class c, where $r = t_1 \ldots t_n$, and $\tau, \tau_1, \ldots, \tau_n$ are expressions of respective types $c, t_1, \ldots, t_n$, then $\tau.m(\tau_1, \ldots, \tau_n)$ is an expression of type t called a *method call.* The expression is interpreted in a state $\mathtt{A}$ by invocation of the corresponding method in such a way that the method associated with the most specific type of the object $\tau^{\mathtt{A}}$ is invoked. In this way, dynamic (late) binding is provided.

It should be noted that both the record projecting operation and the object attribute function are partial functions which may not be defined on some argument values. In this way, null values are modeled. Any predicate evaluated on an undefined argument does not hold, therefore a two-valued logic is used.

3 Example Schema

The following flattened schema for the well-known database "Company" will be used in subsequent examples (methods are omitted).

```
type Name = rec fname: String, minit: Char, lname: String end;

Address = class                     Project = class
 city: String;                        pname: String;
 zip_code: int;                       location: Address;
 state: String                        people: set Employee
end                                 end

Employee = class                    Department = class
 ename: Name;                         dname: String;
 sex: (m, f);                         manager: Employee;
 bdate: Date;                         location: set String;
 ssn: int;                            controls: set Project
 address: Address;                  end
 salary: int;
 supervisor: Employee;              Dependant = class
 works_for: Department;               name: String;
 works_on: set Project;               sex: (m, f)
 dependants: set Dependant          end
end
```

In ODMG some of the above attributes could be defined as relationships. We do not make difference between attributes and relationships in this paper, considering both of them as properties. We also assume that a class extent is denoted by the class name.

4 Object Algebra

In addition to elementary expressions such as attribute access and method call, the object data model includes facilities for constructing more complex expressions representing data retrieval. The set of all possible expressions in an object data model constitutes an *object algebra.* A complex expression normally evaluates to a set or bag that can be formed from one or more argument sets or bags. Having a set of records and a set of objects, we can handle them similarly in many cases, basing on their projecting functions and attribute functions, respectively. We use the term *structure* for both record and object in the sequel. We also use the term *property* for both record fields and object attributes. A short review of the object algebra defined in [8] follows. The interested reader can find equational specifications of data retrieval expressions in Appendix.

4.1 Database Signature and Algebra

A database schema defines a database signature $\Sigma_0 = (T, F)$ where T is a set of data type names and class names and F a set of operation names, attribute names, and method names indexed with their profiles. If an attribute a is declared in a class c as $a : \ t$, then its profile is $c \to t$, and if a method m is declared in c as $m : t_1, ..., t_n \to t$, then its profile is $c, t_1, ..., t_n \to t$. Any particular database state is an algebra of this signature.

Some expressions may evaluate to types that are not defined in the database signature. So, we may speak of a query signature and algebra as respective extensions of the database signature and algebra. In this case, the database signature is extended by some extra type declarations and its algebra is extended by the corresponding types (sorts and operations). If a signature Σ is extended to a signature Σ' and a Σ-algebra $\mathtt{A}$ is extended to a Σ'-algebra $\mathtt{A}'$, we use the index $\mathtt{A}$ to denote those components of $\mathtt{A}'$ that are the same as in $\mathtt{A}$.

4.2 Quantification Expressions

If x is a variable of type t, s an expression of a collection type $col(t)$ and b a Boolean expression, then

`forall` $x : s!b$ and `exists` $x : s!b$

are Boolean expressions. The first expression evaluates to `true` if the collection s is empty or the condition b holds for each element of the collection. The second expression evaluates to `true` if the collection s is nonempty and the condition b holds for at least one element of the collection.

4.3 Mapping Expressions

If $e1$ and $e2$ are Σ-expressions of respective types $set(t_1)$ and $bag(t_1)$ and $f : \ t_1 \to t_2$ a function name in $\Sigma = (T, F)$ such that t_2 is neither a set type nor a bag type, then $e1.f$ is an expression of type $set(t_2)$ and $e2..f$ is an expression of type $bag(t_2)$. The interpretation of these expressions produces a collection by applying the function f to each element of the original collection.

If $e1$ and $e2$ are Σ-expressions of respective types $set(t_1)$ and $bag(t_1)$ and g a function name in Σ with profile either $t_1 \to set(t_2)$ or $t_1 \to bag(t_2)$, then $e1.g$ and $e2..g$ are Σ-expressions of respective types $set(t_2)$ and $bag(t_2)$. The interpretation of these expressions produces a collection by uniting the results of the application of the function g to each element of the original collection.

If a_1 is a component (field, attribute) of a structure s of type c, then $s * a_1 * ... * a_n$, where "*" stands either for "." or for "..", is called a c-path. The a_n is called the *suffix* of the path.

4.4 Projection Expressions

If c is the name of a structure type, x a variable of type c, $p_1, ..., p_k$ identifiers, $e, e_1, ..., e_k$ are $(\Sigma, \{x\})$-expressions of respective types $t, t_1, ..., t_k$ and s a Σ-expression of type $col(c)$, then

$\pi_{p_1:e_1,...,p_k:e_k}(x:s)$ and $\pi_e(x:s)$
are expressions of respective types
$set(\texttt{rec}\ p_1:t_1,...,p_k:t_k\ \texttt{end})$, and $set(t)$
called *projections* of a collection of structures either to the list of expressions $e_1,...,e_k$ or to the expression e. The interpretation of the first expression produces a set of records by evaluating the expressions $e_1,...,e_k$ for each element of s and eliminating duplicate records. The interpretation of the second expression produces a set of elements of type t by evaluating the expressions e for each element of s. Similarly, each of the expressions $\Pi_{p_1:e_1,...,p_k:e_k}(x:s)$ and $\Pi_e(x:s)$ produces a bag of records retaining duplicate records.

4.5 Selection Expression

If c is the name of a structure (record, object) type from the signature $\Sigma = (T,F)$, x a variable of type c, p a $(\Sigma,\{x\})$-expression of type Boolean, and s a Σ-expression of type $set(c)$, then $\sigma_p(x:s)$ is an expression of type $set(c)$, called *selection* of elements of s according to p. The interpretation of the expression produces a set of records by selecting those elements of s that satisfy the condition p. The expression produces a bag if s is a bag.

4.6 Unnesting Join Expression

This expression in fact replaces the *join* operation of the relational algebra because relationships between different sets of objects in the object database are represented by object identifiers rather than by relation keys. Unnesting join of a collection c, such that each its element c_i contains a subcollection c'_i, consists in pairing of c_i with each element of c'_i. It is defined in [8] to allow unnesting join of a hierarchy of nested collections or several independent flat collections. However, according to the Standard [5], several independent hierarchies may be involved in the unnesting process. For this reason, we have to redefine the operation to allow unnesting join of a forest of trees. If e is an expression with variables and ξ a variable assignment, then the notation $(e\xi)^{\mathtt{A}}$ means in the sequel the interpretation of e in algebra $\mathtt{A}$ with the variables bound to the indicated algebra elements.

If $t_1, t_2..., t_n$ are types from the signature $\Sigma = (T,F)$, $X = \{x_1, x_2, ..., x_n\}$ a set of identifiers, e_1 a Σ-expression of type $set(t_1)$, e_2 a $(\Sigma,\{x_1\})$-expression of type $set(t_2)$, ... , e_n a $(\Sigma,\{x_1,...,x_{n-1}\})$-expression of type $set(t_n)$, then
$\mu(x_1:e_1, x_2:e_2, ..., x_n:e_n)$
is an expression of type
$set(\texttt{rec}\ x_1:t_1,\ x_2:t_2,\ ...\ ,\ x_n:t_n\ \texttt{end})$,
called *unnesting join* of $e_1, e_2, ..., e_n$, from the signature
Σ'=($T\cup$ {$\texttt{rec}\ x_1:t_1,\ x_2:t_2,...,x_n:t_n\ \texttt{end}$, $set(\texttt{rec}\ x_1:t_1,\ x_2:t_2,...,x_n:t_n\ \texttt{end})$}, F).
A Σ'-algebra $\mathtt{A}'$ is produced from a Σ-algebra $\mathtt{A}$ by its extension with the types
$\mathtt{A}'_{\mathtt{rec\ x_1:t_1,x_2:t_2,...,x_n:t_n\ end}}$ and $\mathtt{A}'_{\mathtt{set(rec\ x_1:t_1,x_2:t_2,...,x_n:t_n\ end)}}$,
and the expression is interpreted as follows: $[\![\mu(x_1:e_1, x_2:e_2, ..., x_n:e_n)]\!]^{\mathtt{A}'} =$

$\{\mathtt{rec}(\mathtt{v_1, v_2, ..., v_n}) \mid \mathtt{v_1} \in \mathtt{e_1^A},\ \mathtt{v_2} \in (\mathtt{e_2}\xi_1)^{\mathtt{A}}, ...,\ \mathtt{v_n} \in (\mathtt{e_n}\xi_{\mathtt{n-1}})^{\mathtt{A}}\}$, where $\xi_1 = \{\mathtt{x_1} \mapsto \mathtt{v_1}\}, ..., \xi_{n-1} = \{\mathtt{x_1} \mapsto \mathtt{v_1}, ..., \mathtt{x_{n-1}} \mapsto \mathtt{v_{n-1}}\}$ are variable assignments.

Note that each next e_i in the sequence of expressions may use variables $x_1, ..., x_{i-1}$, but is not obliged to use any of them; therefore, we have the usual Cartesian product if the expression involves independent collections.

The expression has type $bag(\mathtt{rec}\ x_1 : t_1,\ x_2 : t_2, ..., x_n : t_n\ \mathtt{end})$ if at least one of e_i has type $bag(t_i)$.

Example. The expression

μ(x: Employee, y: x.works_for, z: y.controls, u: Project, v: u.people)
produces the Cartesian product of two unnested hierarchies, one starting from the set *Employee* and the other one starting from the set *Project.*

4.7 Partitioning Expression

If c is the name of a structure type from the signature $\Sigma = (T, F)$, x a variable of type c, $e_1, ..., e_n$ are $(\Sigma, \{x\})$-expressions of respective types $t_1, ..., t_n$, $p_1, ..., p_n$ identifiers, and s a Σ-expression of type $set(c)$, then

$\rho_{p_1:e_1,...,p_n:e_n}(x : s)$
is an expression of type

$set(\mathtt{rec}\ p_1 : t_1, ..., p_n : t_n, partition : set(c))\ \mathtt{end}$,
called *partitioning* of s according to $e_1, ..., e_n$.

The interpretation of this expression partitions the set s into several groups so that the set *partition* in each group contains elements of s giving the same combination of values of expressions $e_1, ..., e_n$ associated with the fields $p_1, ..., p_n$. The expression evaluates to the result of type $bag(\mathtt{rec}\ p_1 : t_1, ...p_n : t_n,$ $partition : bag(c)\ \mathtt{end})$ when s is a bag (see example in Section 6.3).

5 General Form of an OQL Query

There are two general forms of a OQL query: a *select-from-where-order_by* and *select-from-where-group_by-order_by.* To save space, we will ignore the section *order_by* of both forms. The first form is as follows:

select [distinct] $f(x_1, ..., x_n)$
from x_1 in e_1, x_2 in $e_2(x_1)$, ... , x_m in $e_n(x_1, ..., x_{n-1})$
[where $p(x_1, ..., x_n)$]

and the second one:

select [distinct] $f(y_1, ..., y_m, partition)$
from x_1 in e_1, x_2 in $e_2(x_1)$, ... , x_m in $e_n(x_1, ..., x_{n-1})$
[where $p(x_1, ..., x_n)$]
group by $y_1 : g_1(x_1, ..., x_n), ..., y_m : g_m(x_1, ..., x_n)$
[having $h(y_1, ..., y_m, partition)$] [1],

[1] In fact, in a subquery, each expression e_i and each g_j may contain extra variables $x_{n+1}, \ldots, x_{n+p}$ declared in an enclosing query. However, we will show in the sequel that the identifiers declared as variables in the enclosing query become part of the signature in which the subquery is evaluated. Therefore, without loss of generality we can regard these variables as ground terms of the corresponding signature.

where e_i is a collection of elements of a type t_i and the variable *partition* is bound to the set of all groups, where each group is a record of all x_i values that have the same $y_1, ..., y_n$ values.

Algebraically, we can define the components of the queries as follows. Let Σ be the signature of the query, $col(t)$ denote as always either the type $set(t)$ or $bag(t)$, and $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m, partition\}$ be sets of identifiers such that $X \cap Y = \emptyset$. Then

- e_1 is a Σ-expression of type $col(t_1)$, e_2 a $(\Sigma, \{x_1\})$-expression of type $col(t_2)$, ..., e_n a $(\Sigma, \{x_1, ..., x_{n-1}\})$-expression of type $col(t_n)$,
- p is a (Σ, X)-expression of type Boolean,
- g_1, ..., g_m are (Σ, X)-expressions of respective types t_{11}, ..., t_{1m},
- h is a $(\Sigma, Y \cup \{partition\})$-expression of type Boolean, and
- f is a (Σ, X)-expression of type t in the first case and a $(\Sigma, Y \cup \{partition\})$-expression of type t in the second case.

The type of the query is $col(t)$ where t is the type of the expression f.

We consider in the sequel that omission of the clause `where` or `having` means that the corresponding expression is *true*. The semantics of the query clauses will be represented in the following form: $[\![clause]\!]^{\Sigma,X} = [\![expression]\!]^{\Sigma',Y}$, which means that the semantics of the query *clause* composed in the signature Σ with the set of variables X is equivalent to the semantics of the object algebra *expression* composed in the signature Σ' with the set of variables Y (both X and Y may be empty sets). The semantics of the object algebra expressions is formally defined in [8] and informally explained in Section 4.

6 Semantics of the Query Clauses

6.1 Semantics of the Clause FROM

This clause has the same structure in both forms of the query. So, let Σ be an initial signature, $X = \{x_1, ..., x_n\}$ an initial set of identifiers, e_1 a Σ-expression of type $col(t_1)$, e_2 a $(\Sigma, \{x_1\})$-expression of type $col(t_2)$, ..., e_n a $(\Sigma, \{x_1, ..., x_{n-1}\})$-expression of type $col(t_n)$. Then

$$[\![\texttt{from } x_1 \texttt{ in } e_1,\ x_2 \texttt{ in } e_2(x_1),\ \dots,\ x_m \texttt{ in } e_n(x_1, ..., x_{n-1})]\!]^{\Sigma,X} = [\![\mu(x_1 : e_1, x_2 : e_2, ..., x_n : e_n)]\!]^{\Sigma'}$$

where Σ' is an extension of Σ by the type $col(\texttt{rec } x_1 : t_1,\ x_2 : t_2, ..., x_n : t_n \texttt{ end})$, which is the type of the expression. In this way, we get the direct translation of the clause FROM into the *unnesting join* object algebra expression. Note that $x_1, ..., x_n$ become record field identifiers in the new signature.

Example. The clause FROM of the following query:

```
select struct(empl: x.ename, city: z.city)
from Employee x,
   x.works_on y,
   y.location z
where z.state = "Texas"
```

is translated into the following expression:

$\mu(x : Employee,\ y : x.works\_on,\ z : y.location)$

whose type is

$set(\texttt{rec}\ x :\ Employee,\ y :\ Project,\ z :\ Address\ \texttt{end})$.

If the clause contains only one expression e of type $col(t)$ (the set of identifiers X is a singleton set in this case), then there is no reason for using the operation μ, and the semantics of the clause can be defined in a simpler way:

$[\![\texttt{from}\ x\ \texttt{in}\ e]\!]^{\Sigma,\{x\}} = [\![e]\!]^{\Sigma}$.

Note that $\Sigma' = \Sigma$ in this case, i.e., the signature is not changed.

6.2 Semantics of the Clause WHERE

Let X be the initial set of identifiers, Σ' the signature produced by the interpretation of the clause FROM, s a Σ'-expression of type $col(\texttt{rec}\ x_1 : t_1,\ x_2 : t_2,$ ... , $x_n : t_n\ \texttt{end})$ denoting the collection obtained by the interpretation of the FROM-clause, and p a Σ'-expression of type Boolean, then

$[\![s\ \texttt{where}\ p]\!]^{\Sigma'} = [\![\sigma_{p'}(u : s)]\!]^{\Sigma',\{u\}}$

where u is a variable and p' is a $(\Sigma', \{u\})$-expression obtained by qualifying by u of each occurrence of $x \in \{x_1, ..., x_n\}$ in p.

Example. If s denotes the collection obtained by the interpretation of the FROM-clause of the previous example, then the clause WHERE of the example is translated into the following expression:

$\sigma_{u.z.state="Texas"}(u : s)$

whose type is

$set(\texttt{rec}\ x :\ Employee,\ y :\ Project,\ z :\ Address\ \texttt{end})$.

If $X = \{x\}$ (i.e., the clause FROM contains only one collection expression), then the semantics of the clause is defined in a simpler way:

$[\![s\ \texttt{where}\ p]\!]^{\Sigma,\{x\}} = [\![\sigma_p(x : s)]\!]^{\Sigma,\{x\}}$.

Example. The following part of a query:

```
from Employee e
where e.salary < 1000
```

is translated into the following expression:

$\sigma_{e.salary<1000}(e : Employee)$.

6.3 Semantics of the Clause GROUP_BY

Let X be the initial set of identifiers, Σ' the signature produced by the interpretation of the clause FROM, s a Σ'-expression of type $col(\texttt{rec}\ x_1 : t_1,\ x_2 : t_2,$ ... , $x_n : t_n\ \texttt{end})$ denoting the collection obtained by the interpretation of the WHERE-clause, g_1, ..., g_m are Σ'-expressions, and $Y = y_1, ..., y_m$ a set of variables, then

$[\![s\ \texttt{group by}\ y_1 : g_1, ..., y_m : g_m]\!]^{\Sigma',Y} = [\![\rho_{y_1:g_1',...,y_m:g_m'}(u : s)]\!]^{\Sigma'',\{u\}}$

where u is a variable, Σ'' the signature produced by extension of Σ' by types

$\texttt{rec}\ y_1 : t_{11}, ..., y_m : t_{1m}, partition : set(s)\ \texttt{end}$ and

$col(\texttt{rec}\ y_1 : t_{11}, ..., y_m : t_{1m}, partition : set(s)\ \texttt{end})$, and g'_i a $(\Sigma'', \{u\})$-expression obtained by qualifying by u of each occurrence of $x \in \{x_1, ..., x_n\}$ in g_i.

Example. Suppose we have the following part of the query:

```
from Employee x, x.works_on y, y.location z
where z.state = "Texas"
group by ad:z.city
```

As it is shown in the previous sections, the interpretation of the first two clauses of this query produces a collection, say s, of type

$set(\texttt{rec}\ x : Employee,\ y : Project,\ z : Address\ \texttt{end})$.

The last clause of the query is translated in this case into the following expression:

$\rho_{ad:u.z.city}(u : s)$

whose type is

set(rec ad:string, partition: set(rec x:Employee, y:Project, z:Address end) end).

Once again, if $X = \{x\}$, i.e., the clause FROM contains only one collection expression, then the semantics of the clause is defined in a simpler way:

$$[\![s\ \texttt{group by}\ y_1 : g_1, ..., y_n : g_m]\!]^{\Sigma,\{x\}\cup Y} = [\![\rho_{y_1:g_1,...,y_n:g_n}(x : s)]\!]^{\Sigma'',\{x\}}.$$

Example. The following part of the query from [5], page 114,

```
from Employee e
group by low: e.salary<1000
         medium: e.salary>=1000 and e.salary<10000
         high: e.salary>=10000
```

is translated into the following expression:

$\rho_{low:e.salary<1000,medium:e.salary>=1000\&e.salary<10000,high:e.salary>=10000}(e : s)$.

This gives a set of three elements of type

set(rec low:Boolean, medium:Boolean, high:Boolean, partition:set(Employee) end).

6.4 Semantics of the Clause HAVING

Let Σ' be the signature produced by the interpretation of the clause GROUP_BY, s a Σ'-expression of type $col(\texttt{rec}\ y_1 : t_{11}, ..., y_m : t_{1m}, partition : set(s)\ \texttt{end})$ denoting the collection obtained by the interpretation of the clause as defined above, and h a (Σ')-expression of type Boolean, then

$$[\![s\ \texttt{having}\ h]\!]^{\Sigma'} = \sigma_{h'}(u : s)^{\Sigma',\{u\}}$$

where u is a variable and p' is a $(\Sigma', \{u\})$-expression obtained by qualifying by u of each occurrence of $y \in \{y_1, ..., y_m, partition\}$ in h.

Example. Consider the following part of the query from [5], page 114,

```
from Employee e
group by department: e.dname
having avg(select x.salary from partition x) > 30000 ².
```

The interpretation of the first two lines of the query produces a collection s of

[2] In page 114 of [5] the clause HAVING is written in the following way: `having avg(select x.e.salary from partition x) > 30000`. This is due to the fact that the Standard considers the semantics of the clause FROM to be a bag of records even when the clause contains only one collection (i.e., the record type consists only of one field, with name e in this case). We simplify the syntax a little bit for this special case, allowing the semantics of the clause to be the collection indicated.

type $set($`rec` $department: string, partition: set(Employee)$ `end`). Then the clause HAVING of the example is translated into the following expression:

$\sigma_{avg(\texttt{select x.salary from u.partition x})>30000}(u:s)$.

6.5 Semantics of the Clause SELECT

Generally, the syntax of the SELECT clause is the following one:
`select` [`distinct`] `*` or
`select` [`distinct`] *projection* {*,projection*}
where *projection* is one of the following forms:
1) *expression* as *identifier*
2) *identifier* : *expression*
3) *expression*

This is an alternative syntax for
`struct` $(identifier:\ expression\ \{,\ identifier:\ expression\})$,
which will be used in the sequel.

The interpretation of the clause always produces a collection (a set if `distinct` is present, a bag otherwise) of elements of type t, where t is constructed as follows:

- it is the type of the elements of the collection produced by the rest of the query if the option "`*`" is used;
- it is the type of the expression e if there is only one *projection* and syntax (3) is used; otherwise:
- it is a record type where each *identifier* is a field identifier and the type of a field is defined by the type of the corresponding expression. If an expression is not supplied by an identifier, then it must be a path (possibly of length one) ending with a property (suffix of the path) whose name is then chosen as the identifier. We will use this identifier explicitly in examples.

There is also syntactic sugar in using aggregate operations to provide compatibility with the SQL syntax. If *aggregate* stands for one of *min*, *max*, *sum*, and *avg*, then

- `select count(*) from ...` is equivalent to `count(select * from ...)`,
- `select` *`aggregate`*`(query) from ...` is equivalent to *`aggregate`* `(select query from ... )`.

We will consider only canonical forms in the sequel. Several forms of the SELECT clause will be discussed separately.

1. Let Σ' be the signature of the rest of the query and s denote the collection produced by the interpretation of the rest of the query, then:
$[\![\texttt{select} * s]\!]^{\Sigma'} = [\![s]\!]^{\Sigma'}$ and $[\![\texttt{select distinct} * s]\!]^{\Sigma'} = [\![s']\!]^{\Sigma'}$,
where $s' = s$ if s is a set and $s' = setof(s)$ if s is a bag.

Example. The following query:

```
select *
from Employee x,
   x.works_on y,
   y.location z
where z.state = "Texas"
```

is translated into the following expression:

$\sigma_{u.z.state="Texas"}(u : \mu(x : Employee, y : x.works\_on, z : y.location))$

producing a result of type

$set(\texttt{rec}\ x : Employee,\ y : Project,\ z : Address\ \texttt{end})$.

2. Let Z denote the set of variables X in the first form of the query and the set $Y \cup \{partition\}$ in the second form, Σ' be the signature of the rest of the query, e an expression (a path) of type t, and s denote the collection produced by the interpretation of the rest of the query, then:

$[\![\texttt{select}\ e\ s]\!]^{\Sigma',Z} = [\![\Pi_{e'}(u : s]\!]^{\Sigma',\{u\}}$ and

$[\![\texttt{select distinct}\ e\ s]\!]^{\Sigma',Z} = [\![\pi_{e'}(u : s]\!]^{\Sigma',\{u\}}$ where u is a fresh variable and e' is obtained from e by qualifying by u of each occurrence of $z \in Z$ in e.

Example. The query:

```
select distinct e.ename
from Department d
     d.controles c
     c.people e
```

is translated into the following expression:

$\pi_{u.e.ename}(u : \mu(d : Department,\ c : d.controls,\ e : c.people))$

producing a result of type $set(Name)$.

Once again, if $Z = X = \{\text{x}\}$, (i.e., only one collection is mentioned in the clause FROM and there is no GROUP_BY clause), then translation is performed in a simpler way:

$[\![\texttt{select}\ e\ s]\!]^{\Sigma',\{x\}} = [\![\Pi_e(x : s]\!]^{\Sigma',\{x\}}$ and

$[\![\texttt{select distinct}\ e\ s]\!]^{\Sigma',\{x\}} = [\![\pi_e(x : s]\!]^{\Sigma',\{x\}}$.

Example. The query:

```
select e.ename from Employee e
```

will be translated into the following expression:

$\Pi_{e.ename}(e : Employee)$.

3. Let Z denote the set of variables X in the first form of the query and the set $Y \cup \{partition\}$ in the second form, Σ' be the signature of the rest of the query, $p_1, ..., p_k$ identifiers, $e_1, ..., e_k$ expressions of respective types $t_{21}, ..., t_{2k}$, and s denote the collection produced by the interpretation of the rest of the query, then:

$[\![\texttt{select}\ p_1 : e_1, ..., p_k : e_k\ s]\!]^{\Sigma',Z} = [\![\Pi_{p_1:e'_1,...,p_k:e'_n}(u : s)]\!]^{\Sigma'',\{u\}}$

where u is a variable, Σ'' is obtained by extending Σ' with the type $bag(\texttt{rec}\ p_1 : t_{21}, ..., p_k : t_{2k}\ \texttt{end})$, e'_i is obtained from e_i by qualifying by u of each occurrence of $z \in Z$ in e_i. A result of type $bag(\texttt{rec}\ p_1 : t_{21}, ..., p_k : t_{2k}\ \texttt{end})$ is produced.

If the clause contains the word `distinct`, then
$[\![\texttt{select distinct } p_1 : e_1, ..., p_k : e_k\ s]\!]^{\Sigma',Z} = [\![\pi_{p_1:e'_1,...,p_k:e'_n}(u : s)]\!]^{\Sigma'',\{u\}}$,
where Σ'' is obtained by extending Σ' with the result type $set(\texttt{rec}\ p_1 : t_{21}, ..., p_k : t_{2k}\ \texttt{end})$.

As in the previous case, if $Z = X = \{\text{x}\}$, (i.e., only one collection is mentioned in the clause FROM and there is no GROUP_BY clause), then translation is performed in a simpler way:
$[\![\texttt{select } p_1 : e_1, ..., p_k : e_k\ s]\!]^{\Sigma',\{x\}} = [\![\Pi_{p_1:e_1,...,p_k:e_n}(x : s)]\!]^{\Sigma'',\{x\}}$ and
$[\![\texttt{select distinct } p_1 : e_1, ..., p_k : e_k\ s]\!]^{\Sigma',\{x\}} = [\![\pi_{p_1:e_1,...,p_k:e_n}(x : s)]\!]^{\Sigma'',\{x\}}$,

Example. Let us discuss step by step the translation of the following query:

```
select dep: department,
       avg_salary: avg(select x.salary from partition x)
from Employee e
group by department: e.works_for.dname
having avg(select x.salary from partition x) > 30000.
```

1. The clause FROM is translated in the database signature Σ_0 with the set of variables $X = \{e\}$, and the result is a Σ_0-expression *Employee*.

2. The clause GROUP_BY is translated in the same signature with the set of variables $X = \{e\}$ and $Y = \{department, partition\}$ into the expression
$s1 = \rho_{department:e.works\_for.dname}(e : Employee)$
of type $set(\texttt{rec}\ department : string,\ partition : set(Employee)\ \texttt{end})$ extending the signature Σ_0 to a signature Σ_1.

3. The clause HAVING is first translated into a Σ_1-expression
$\sigma_{avg(select\ x.salary\ from\ u1.partition\ x)\ >\ 30000}(u1 : s1)$.
This expression contains a subquery
`select x.salary from u1.partition x`.
The starting signature for this query is Σ_1 with the set of variables $X = \{x\}$.

3.1. The translation of the FROM clause of this query produces a Σ_1-expression $u1.partition$ of type $set(Employee)$.

3.2. The clause SELECT of this query is translated in the signature Σ_1 with the set of variables $Z = \{x\}$, which finally produces a Σ_1-expression
$s2 = \Pi_{x.salary}(x : u1.partition)$
of type *int*. As a result, the whole clause HAVING is translated into a Σ_1-expression
$s3 = \sigma_{avg(s2)>30000}(u1 : s1)$
of type $set(\texttt{rec}\ department : string,\ partition : set(Employee)\ \texttt{end})$.

4. The clause SELECT of the main query is translated in the signature obtained by the translation of the clause GROUP_BY (i.e, the signature Σ_1) with the set of variables $Z = \{department, partition\}$ producing the expression
$\Pi_{dep:\ u2.department,\ avg\_salary:\ avg(select\ x.salary\ from\ u2.partition\ x)}(u2 :\ s3)$,
of type $bag(\texttt{rec}\ dep : string,\ avg\_salary : int\ \texttt{end})$ extending the signature Σ_1 to a signature Σ_2. This expression also contains a subquery
`select x.salary from u1.partition x`.
In the same way as it is explained in steps 3.1-3.2, it is translated into a Σ_2-

expression

$s4 = \Pi_{x.salary}(x : u2.partition)$

of type *int*. As a result the whole query is translated into the expression

$\Pi_{dep:\ u2.department,\ avg\_salary:avg(s4)}(u2 :\ s3).$

7 Related Work and Conclusion

There are many papers proposing object algebra for certain object-oriented data models. One of the first work in this field is presented in [4]. The approach uses high-order operations basing on some kind of high-level algebra not defined in the paper. The algebra is not used for defining semantics of a query language. The object algebras presented in [6,2] are also aimed at providing data manipulation facilities for certain data models rather than giving semantics of a query language. The most interesting work in this field is presented in [1] where monoid comprehension calculus is used as an intermediate level between OQL and an object algebra that is also defined in terms of monoid calculus. The approach is not based on any formal database model and there is no indication in what way object algebra can be interpreted in a particular database state. In contrast to this work, we do not use an intermediate level and map OQL queries directly into object algebra expressions whose semantics is defined in terms of state algebra. One can find a more detailed review of the above (and some other works) in [8] and their critique in [7].

Thus, we have formally defined the semantics of OQL in terms of object algebra and have shown in multiple examples that OQL queries can be easily expressed by means of object algebra expressions. As a result, an OQL query can be mechanically translated into the corresponding object algebra expression, which can be further optimized and executed. We have not yet elaborated an optimization technique for the object algebra proposed. This remains a subject of further work.

References

1. L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *ACM Transactions on Database Systems*, December 2000.
2. K. Lellahi. Modeling data and objects: An algebraic viewpoint. *Proc. 1st summer school in theoretical aspects of computer science*, Tehran-Iran, July 2000 (to appear in LNCS).
3. K. Lellahi and A. Zamulin. Object-Oriented Database as a Dynamic System With Implicit State. *Advances in Databases and Information Systems (Proceedings of the 5th East European Conference, ADBIS 2001, Vilnus, Lithuania, September 2001)*, LNCS, vol. 2151, pp. 239–252.
4. T.W. Leung, G. Mitchell, B. Subramanian, et el. The AQUA Data Model and Algebra. *Proc. 4th Workshop on Database Programming Languages*, Springer Workshops in Computing, 1993, pp. 157–175.
5. *The Object data Standard ODMG 3.0.* Morgan Kaufmann Publishers, 2000.

6. I. Savnik, Z. Tari, T. Mohoric. QAL: A Query Algebra of Complex Objects. *Data & Knowledge Eng. Journal, North-Holland*, Vol.30, No.1, 1999, pp.57–94.
7. K. Subieta and J. Leszczylowski. A Critique of Object Algebras. http://www.ipipan.waw.pl/ subieta/EngPapers/CritiqObjAlg.html
8. A.V. Zamulin. An Object Algebra for the ODMG Standard. *Y. Manolopoulos and P. Navrat (eds.). Advances in Databases and Information Systems (Proceedings of the 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, September 2002)*, LNCS, vol. 2435, pp. 291–304.

Appendix. Equational Specification of Object Algebra

It is assumed that both the set type and the bag type are equipped with the operations *empty*, *insert* and $\cup$ with conventional semantics, and any set (bag) value can be built by the operations *empty* and *insert*. and bags in such a way that if $s_1, ..., s_n$ are (sets or bags) of respective types $col(t_1), ..., col(t_2)$, then type

An equation consists of two parts separated by the equivalence metasymbol "==". Universally quantified variables used in equations are declared before each equation. The type $col(t)$ in a variable declaration means either $set(t)$ or $bag(t)$ (i.e. the equation holds for both sets and bags). The operations are sometimes qualified by the type name (using the syntax of C++) to clearly indicate the type they belong to. Conventional conditional expressions and the definedness predicate `D` (permitting one to check the definedness of an expression) are used in some equations.

If e is an expression containing a variable x and y a variable different from x, then the notation $e[y/x]$ means substitution of x by y in e. Any expression belongs to the signature $\Sigma' = (T', F)$ extending either the database signature or the signature of the enclosing expression by new types and is interpreted in a Σ'-algebra A'. An algebra satisfies an equation if ether both parts of the equation are defined and evaluates to the same algebra element or both are undefined.

`1. Quantification expressions`

$\forall y : t, s : col(t), b : Boolean.$
`forall` $x : empty!b == true;$
`forall` $x : insert(y, s)!b == b[y/x] \wedge$ `forall` $x : s!b;$

`exists` $x : empty!b == false;$
`exists` $x : insert(y, s)!b == b[y/x] \vee$ `exists` $x : s!b;$

`2. Mapping and projection expressions`

Let $f : t_1 \to t_2$ and $g : t_1 \to set(t_2)$ be function names from F. Then:
$\forall e1 : set(t_1), y : t_1.$
$set(t_1) :: empty.f == set(t_2) :: empty;$
$set(t_1) :: insert(y, e1).f ==$ `if` $\mathtt{D}(f(y))$ `then` $set(t_2) ::$
$insert(f(y), e_1.f)$ `else` $e_1.f;$

$set(t_1) :: empty.g == set(t_2) :: empty;$
$set(t_1) :: insert(y, e1).g ==$ `if` $\mathtt{D}(g(y))$ `then` $set(t_2) ::$
$union(g(y), e_1.g)$ `else` $e_1.g;$

The equations for bag mapping are defined in a similar way.

Let $R = \texttt{rec}\ p_1 : t_1, ..., p_k : t_k\ \texttt{end}$ be a type from T' and $e, e_1, ..., e_k$ expressions of respective types $t, t_1, ..., t_k$. Then:
$\forall\ s :\ col(c),\ y :\ c,\ s' :\ col(t),\ y' :\ t.$
$\pi_{p_1:e_1,...,p_k:e_k}(x : col(c) :: empty) == set(R) :: empty;$
$\pi_{p_1:e_1,...,p_k:e_k}(x : col(c) :: insert(y, s)) ==$
$\quad set(R) :: insert(rec(e_1[y/x], ..., e_k[y/x]), \pi_{p_1:e_1,...,p_k:e_k}(x : s);$

$\pi_e(x : col(t) :: empty) == set(t) :: empty;$
$\pi_e(x : col(t) :: insert(y', s')) == set(t) :: insert(e[y'/x]), \pi_e(x : s');$

The operation Π is defined in a similar way.

`3. Selection expression`

Let p be a Boolean expression involving the variable x. Then:
$\forall s : col(c), y : c.$
$\sigma_p(x : col(c) :: empty) == col(c) :: empty;$
$\sigma_p(x : col(c) :: insert(y, s)) ==$ `if` $p[y/x]$ `then` $col(t) ::$
$insert(y, \sigma_p(s))$ `else` $\sigma_p(s);$

`4. Unnesting join expression`

Let $R = \texttt{rec}\ x_1 : t_1, x_2 : t_2, \ldots, x_n : t_n\ \texttt{end}$ be a type from T', $x_1, x_2, \ldots, x_n$ identifiers, e_1 a Σ-expression of type $col(t_1)$, e_2 a $(\Sigma, \{x_1\})$-expression of type $col(t_n)$, ... , e_n a $(\Sigma, \{x_1, \ldots, x_{n-1}\})$-expression of type $col(t_n)$ (for simplicity, we consider that none of the collections is empty). Then:
$\forall s_1 : col(t_1), s_2 : col(t_2), \ldots, s_n : col(t_n), y_1 : t_1, y_2 : t_2, \ldots, y_n : t_n.$
$\mu(x_1 : e_1, x_2 : e_2, \ldots, x_n : e_n) ==$ let $e_1 = insert(y_1, s_1),\ e_2[y_1/x_1] = insert(y_2, s_2),$
$\ldots, e_n[y_1/x_1, y_2/x_2, y_{n-1}/x_{n-1}] = insert(y_n, s_n)$ in
$\quad \{(y_1, y_2, \ldots, y_n)\} \cup \mu(x_1 : e_1, x_2 : e_2 \ldots, x_n : s_n)) \cup \ldots \cup$
$\quad \mu(x_1 : e_1, x_2 : s_2, \ldots, x_n : e_n) \cup \mu(x_1 : s_1, x_2 : e_2, \ldots, x_n : e_n);$

The expression evaluates to a set of records if all involved collections are sets.

`5. Partitioning expression`

Let $R = \texttt{rec}\ p_1 : t_1, ...p_n : t_n,\ partition : set(c)\ \texttt{end}$ be a type from T' and $e_1, ..., e_n$ expressions, involving the variable x, of respective types $t_1, ..., t_n$. We first define an auxiliary partitioning expression $\rho 1$ using two collections of the same type:

$\forall s, s1 : col(c), y : c.$
$\rho 1_{p_1:e_1,...,p_n:e_n}(x :\ col(c) :: empty, x :\ s1) == col(R) :: empty;$
$\rho 1_{p_1:e_1,...,p_n:e_n}(x :\ col(c) :: insert(y, s), x :\ s1) == col(R)::$
$\quad insert(rec(e_1[y/x], ..., e_n[y/x], \sigma_{e_1[y/x]=e_1,...,e_n[y/x]=e_n}(x :\ s1)),$
$\qquad \rho 1_{p_1:e_1,...,p_n:e_n]}(x :\ s,\ x :\ s1));$

Now we can define the main expression:

$\forall s : col(c).\ \rho_{p_1:e_1,...,p_n:e_n}(x :\ s) = \rho 1_{p_1:e_1,...,p_n:e_n}(x :\ s, x :\ s);$

Similar Sub-trajectory Retrieval for Moving Objects in Spatio-temporal Databases

Choon-Bo Shim and Jae-Woo Chang

Dept. of Computer Engineering,
Research Center of Industrial Technology, Engineering Research Institute
Chonbuk National University, Jeonju, Jeonbuk 561-756, South Korea
{cbsim,jwchang}@dblab.chonbuk.ac.kr

Abstract. Moving objects' trajectories play an important role in doing efficient retrieval in spatial-temporal databases. In this paper, we propose a spatio-temporal representation scheme for modeling the trajectory of moving objects. Our spatio-temporal representation scheme effectively describes not only the single trajectory of a moving object but also the multiple trajectories of two or more moving objects. For measuring similarity between two trajectories, we propose a new k-warping distance algorithm which enhances the existing time warping distance algorithm by permitting up to k replications for an arbitrary motion of a query trajectory. Our k-warping distance algorithm provides an approximate matching between two trajectories as well as an exact matching between them. Based on our k-warping distance algorithm, we also present a similarity measure scheme for both the single trajectory and the multiple trajectories in spatio-temporal databases. Finally, we show from our experiment that our similarity measure scheme based on the k-warping distance outperforms Li's one (no-warping) and Shan's one (infinite-warping) in terms of precision and recall measures.

1 Introduction

Due to the wide spreading of mobile communication facilities such as PCS, PDA, and CNS (Car Navigation System) as well as the rapid increment of sport video data such as soccer, hockey, and basket, it is required to deal with them. Recently, a lot of researches have been done on efficient management and storing for the various information of moving objects, especially being an important research topic in the field of video analysis and video database applications [1, 2]. Meanwhile, the trajectory of moving objects can be represented as a spatio-temporal relationship combining its spatial and temporal property, which plays an important role in doing video analysis & retrieval in video databases [3, 4]. A typical trajectory-based user query on the spatio-temporal relationship is as follows: *"Finds all objects whose motion trajectory is similar to the trajectory shown in a user interface window"*. A moving object is defined as a salient object that is continuously changing at its spatial location over the time. The trajectory of a moving object is defined as a collection of successive motions, each being represented as a spatio-temporal relationship. The moving objects' trajectories are the subject of concern by a user in video databases. Similar sub-trajectory retrieval means searching for sub-trajectories in data trajectories which are similar to a given query trajectory.

In this paper, we first propose a spatio-temporal representation scheme for modeling the trajectory of moving objects. Our spatio-temporal representation scheme effectively describes not only the single trajectory of only one moving object but also the multiple trajectories of two or more moving objects. For measuring similarity between two trajectories, we propose a new k-warping distance algorithm which calculates a k-warping distance between a given query trajectory and a data trajectory by permitting up to k replications for an arbitrary motion of a query trajectory. Our k-warping distance algorithm provides an approximate matching between two trajectories as well as an exact matching between them. Especially in case of a large amount of moving objects, the approximate matching can improve the performance of retrieval on moving objects' trajectories, compared with the exact matching. Based on our k-warping distance algorithm, we finally present a similarity measure scheme for both the single trajectory and the multiple trajectories in spatio-temporal databases.

This paper is organized as follows. In Section 2, we introduce related researches on similar sub-trajectory retrieval. In Section 3, we propose a spatio-temporal representation scheme for modeling the trajectory of moving objects. In Section 4, we propose a new k-warping distance algorithm and present a similarity measure scheme for both the single trajectory and the multiple trajectories. In Section 5, we provide the performance analysis of our similarity measure scheme and compare its performance with those of the existing related researches. Finally, we draw our conclusions and suggest future work in Section 6.

2 Related Work

There have been two main researches on retrieval based on similar sub-trajectory by measuring the similarity between a given query trajectory and data trajectories, i.e., Li's scheme and Shan's scheme. First, Li et al. [3, 4] represented the trajectory of a moving object as eight directions, such as North(NT), Northwest(NW), Northeast(NE), West(WT), Southwest(SW), East(ET), Southeast(SE), and Southwest(SW). They represented as (S_i, d_i, I_i) the trajectory of a moving object A over a given time interval I_i where S_i is the displacement of A and d_i is a direction. For a set of time interval $<I_1, I_2, \ldots, I_n>$, the trajectories of A can be represented as a list of motions, like $<(S_1, d_1, I_1), (S_2, d_2, I_2), \ldots, (S_n, d_n, I_n)>$. Based on the representation for moving objects' trajectories, they present a similarity measures to computes the similarity of spatio-temporal relationships between two moving object. Let $\{M_1, M_2, \ldots, M_m\}$ $(m \geq 1)$ be the trajectory of moving object A, $\{N_1, N_2, \ldots, N_n\}$ be the trajectory of moving object B, and $m \leq n$. The similarity measure between the trajectory of object A and that of object B, TrajSim(A, B), is computed by using the similarity distances of directional relations as follows. Here, minDiff(A, B) and maxDiff(A, B) are the smallest distance between A and B and the largest distance, respectively.

$$TrajSim(A,B) = \frac{\max Diff(A,B) - \min Diff(A,B)}{\max Diff(A,B)} \qquad (\forall j, 0 \leq j \leq n-m)$$

Secondly, Shan and Lee [5] represented the trajectory of a moving object as a sequence of segments, each being expressed as the slope with real angle ranging from 0 to 360 degree for content-based retrieval. They also proposed two similarity measure algorithms, OCM (Optimal Consecutive Mapping) and OCMR (Optimal Consecutive Mapping with Replication), which can measure similarity between query trajectory $Q=(q_1, q_2, \ldots, q_M)$ and data trajectory $V=(v_1, v_2, \ldots, v_N)$. The OCM algorithm that supports exact matching measures the similarity for one-to-one segment mapping between query trajectory and data trajectory. Meanwhile, The OCMR algorithm supports approximation matching. In order to measure the similarity, each motion of query trajectory can be permitted to map with more than one motions of data trajectory. Figure 1 shows the relation of OCMR.

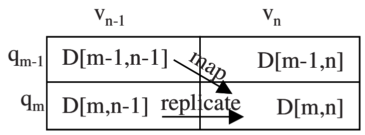


Fig. 1. Relation of OCMR between D[m, n] and D[i, j]

Here, $d(q_i, v_j)$ means distance function which computes distance between q_i and v_j and D[M, N] is minimum cost table which used to measure the minimum distance with replication between query trajectory and data trajectory. There are two possible relations between D[m, n] and D[i, j] for some combinations of smaller i-th and j-th. That is, map means that the motion v_n is mapped with the motion q_m, $D[m,n] = D[m-1,n-1]+d(q_m, v_n)$ and replication means that the motion v_n is replicated to mapped with the motion q_m, $D[m, n]=D[m-1,n]+d(q_m,v_n)$.

3 Spatio-temporal Representation Scheme

Moving objects are salient objects which are continuously changing its locations over time interval. To effectively deal with moving objects, it is necessary to consider both spatial and temporal relationships. Existing schemes, like Li's scheme and Shan's scheme, consider only directional and topological information for modeling the trajectory of moving objects. But, our representation scheme takes into account distance information in addition. The trajectory is defined as a set of motions of moving objects over time interval. We will define a spatio-temporal representation scheme for the trajectory of moving objects, which effectively describes not only the single trajectory of only one moving object but also the multiple trajectories of two or more moving objects.

3.1 Representation for Single Trajectory (ST)

[Definition 1] Motion property information for a moving object A over all the time intervals, MPS(A), is defined as follows:

$$\mathbf{MPS(A) = \{M_i(A) \mid i = 0, \ldots, n-1\} = \{M_0(A), M_1(A), \ldots, M_{n-1}(A)\}}$$

[Definition 2] A motion property for a moving object A over time interval I_i, $M_i(A)$, is defined as follows:

$$\mathbf{M_i(A) = (R_i(A), D_i(A), I_i(A))}$$

Here, $R_i(A)$ is a moving direction over time interval I_i ($=[t_i,t_{i+1}]$) and is represented as a real angle with a range of 0 to 360 degree. $D_i(A)$ is a moving distance over I_i and is described as an absolute Euclidean distance or a relative distance. $I_i(A)$ means a time interval from the start time to the end time while the moving object A is moving.

[Definition 3] Stationary property information for a moving object A over all the time instances, SPS(A), is defined as follows:

$$\mathbf{SPS(A) = \{S_i(A) \mid i = 0, \ldots, n\} = \{S_0(A), S_1(A), \ldots, S_n(A)\}}$$

[Definition 4] A stationary property for a moving object A at time t_i, $S_i(A)$, is defined as follows:

$$\mathbf{S_i(A) = ([L_i(A)], [O_i(A)])}$$

Here, $L_i(A)$ is a location information of the moving object A. The location information describes a real location in coordinates or a semantic-based location according to a real application, e.g., penalty area or goal area in the soccer game. $O_i(A)$ is an object information related with the moving object A, e.g., actor or owner having the moving object A. By [], $L_i(A)$ and $O_i(A)$ are optional properties.

For the single trajectory of a moving object A, it is possible to combine a motion property (Definition 1 and 2) with a stationary property (Definition 3 and 4) as shown in Figure 2. The single trajectory is defined as follows.

[Definition 5] For a given ordered list of time interval $I_0, I_1, \ldots, I_{n-1}$, the single trajectory information of a moving object A, ST(A), is defined as follows:

$$\mathbf{ST(A) = MPS(A) + SPS(A)}$$

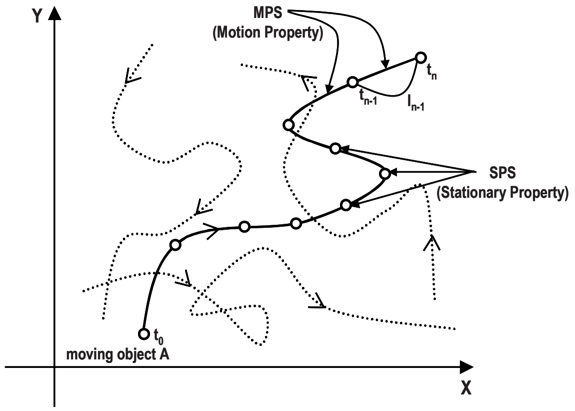


Fig. 2. Representation for the single trajectory of a moving object A

3.2 Representation for Multiple Trajectories (MT)

We define multiple trajectories as the trajectories of two or more moving objects. However, since the multiple trajectories can be represented by the combination of the trajectory between two moving objects, we first define a relationship trajectory (RT) between two objects.

[Definition 6] Let at least one of object A and object B be a moving object. Motion property information for A and B over all the time interval, MPM(A, B), is defined as follows:

$$\mathbf{MPM(A, B) = \{M_i(A, B) \mid i = 0, \dots, n\text{-}1\} = \{M_0(A, B), M_1(A, B), \dots, M_{n-1}(A, B)\}}$$

[Definition 7] Let at least one of object A and object B be a moving object. A motion property for A and B over time interval I_i ($[t_i, t_{i+1}]$), M_i(A, B), is defined as follows:

$$\mathbf{M_i(A, B) = (D_i(A, B), I_i(A, B))}$$

Here, D_i(A, B) is a relative moving distance of A to B over I_i and is ranged from 0 to 100. That is, D_i(A, B) is 50 in case the moving distance of A is the same as that of B. D_i(A, B) is ranged from 51 to 100 in case the moving distance of A is greater than that of B while it is near to 0 as the moving distance of A is less than that of B. I_i(A, B) is the same as single trajectory.

[Definition 8] Let at least one of object A and object B be a moving object. Stationary property information for A and B over all the time instances, SPM(A, B), is defined as follows:

$$\mathbf{SPM(A, B) = \{S_i(A, B) \mid i = 0, \dots, n\} = \{S_0(A, B), S_1(A, B), \dots, S_n(A, B)\}}$$

[Definition 9] Let at least one of object A and object B be a moving object. A stationary property for A and B at time t_i, S_i(A, B), is defined as follows:

$$\mathbf{S_i(A, B) = ([L_i(A)], [O_i(A)], ([L_i(B)], [O_i(B)], T_i(A, B), R_i(A, B))}$$

Here, L_i(A) and L_i(B) are the location information of moving object A and B, respectively. O_i(A) and O_i(B) are the actors having moving objects A and B, respectively. T_i(A, B) is a spatial (topological) relations on XY-coordinates from A to B, being represented as one of seven topological relations operator : FA(FarAway), DJ(DisJoint), ME(MEet), OL(OverLap), CL(is-inCLuded-by), IN(INclude), and SA(SAme). Finally, R_i(A, B) means a directional relations from A to B and is ranged from 0 to 360 degree.

For a relationship trajectory between A and B, it is possible to combine a motion property (Definition 6 and 7) with a stationary property (Definition 8 and 9). The relationship trajectory information is defined as follows.

[Definition 10] Let at least one of object A and object B be a moving object. For a given ordered list of time interval $I_0, I_1, \dots, I_{n-1}$, the relationship trajectory information between A and B, RT(A, B), is defined as follows:

$$\mathbf{RT(A, B) = MPM(A, B) + SPM(A, B)}$$

Based on Definition 5 and 10, the multiple trajectory information of two or more moving objects, MT($A_1, A_2, \dots, A_n$), can be represented by a combination of the relationship trajectory information (RT) and the single trajectory information (ST).

[Definition 11] Among objects $A_1, A_2, \dots, A_n$, let i be the number of moving objects and j be the number of stationary objects, i.e., n=i+j. The multiple trajectory information of $A_1, A_2, \dots, A_n$, MT($A_1, A_2, \dots, A_n$), is defined as follows:

$$MT(A_1, A_2, \dots, A_n) = \{ST(A_p) \mid \mathrm{p} = 1, \dots, \mathrm{i}\} + \{\mathrm{RT}(\mathrm{A_q}, \mathrm{A_{q+1}}) \mid \mathrm{q} = 1, \dots, \mathrm{k}\} \qquad , k = {}_nC_2 - {}_jC_2$$

Here ST(A_i) is the single trajectory information of an object A_i. RT(A_k, A_{k+1}) is the relationship trajectory information between object A_k and A_{k+1} where k is the number of relationship trajectories between two moving objects as well as between a moving object and a stationary object.

Figure 3 shows an example of the multiple trajectory information of three objects: Car(C), Building(B), and Motorcycle(M). The Car object and the Motorcycle object are moving objects (i=2) and the Building object is a stationary object (j=1). Thus, k is 3 and MT(C, M, B) = {ST(C), ST(M), ST(B)} + {RT(C, M), RT(C, B), RT(M, B)}.

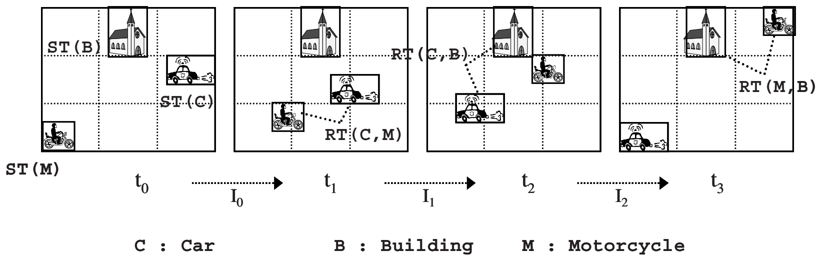


Fig. 3. Multiple trajectory information of Car, Building, and Motorcycle

4 Similarity Measure Scheme for Moving Objects' Trajectories

4.1 k-Warping Distance Algorithms

We first present three considerations for supporting efficient similar sub-trajectory retrieval on moving objects.

<u>**Consideration 1**</u>: The time warping transformation used for a similar sub-sequence matching in sequence databases can allow the infinitive replication of a data sequence as well as a query sequence. However, for similar sub-trajectory retrieval in spatio-temporal databases, it is necessary to allow the replication of only a query trajectory.

<u>**Consideration 2**</u>: The time warping transformation for a similar sub-sequence matching can allow the infinitive replication of an arbitrary motion. However, for the similar sub-trajectory retrieval, it is necessary to support the replication of up to the fixed number (k) of motions, so called k-warping distance.

<u>**Consideration 3**</u>: For modeling motions being composed of the trajectory of a moving object, it is necessary to support multiple properties including angle, distance, and time, instead of the single property of angle.

The consideration 1 is generally needed for supporting an approximation matching from similar sub-trajectory retrieval and the considerations 2 and 3 are needed for improving the effectiveness of the approximation matching. In addition, the considerations 2 and 3 are very sensitive, depending on application areas. The similar subsequence matching approach which is used for the existing time warping transformation does not satisfy all of the above three considerations. The reason is why the characteristic of data used in sequence database is different from that of trajectory data of moving objects in spatio-temporal databases. Generally, the sequence data has a detailed and elaborate feature and the number of elements consisting of a sequence reaches scores or hundreds. On the other hand, the trajectory data of moving objects in spatio-temporal databases are composed of motions over a time interval and the number of motions consisting of a trajectory is less than scores. Meanwhile, the Shan's OCMR scheme can satisfy the considerations 1, but it does not satisfy the considerations 2 and 3.

Therefore, we propose a new k-warping distance algorithm which can support an approximation matching and satisfy the above three considerations for similar sub-trajectory retrieval. In order to satisfy the consideration 3, we generally define the trajectory of moving objects as a collection of consecutive motions consisting of n-dimensional properties.

[Definition 12] The trajectory of moving object S is defined as a set of consecutive motions, S= (s[1], s[2], ..., s[|S|]), where each motion s[i] is composed of n-dimensional properties as follows:

$$\mathbf{s[i] = (s[i, 1], s[i, 2], \ldots, s[i, n])}$$

For measuring a similarity between two trajectories, we define a k-warping distance as follows, which is newly made by applying the concept of time warping distance used for time-series databases to the trajectory data of moving objects in spatio-temporal databases.

[Definition 13] Given two trajectory of moving objects S and Q, the k-warping distance D_{kw} is defined recursively as follows:

$$\mathbf{D_{kw}(0, 0) = 0, D_{kw}(S, 0) = D_{kw}(0, Q) = \infty}$$
$$\mathbf{D_{kw}(S, Q) = D_{base}(S[1], Q[1])+\min(\{D_{kw}((S[2+i:-], Q), 0 \le i \le k), D_{kw}(S[2:-], Q[2:-])\})}$$
$$\mathbf{D_{base}(a, b) = d_{df}(a, b)}$$

Our k-warping distance algorithm is shown in Figure 4. It calculates a k-warping distance between a given query trajectory Q and a data trajectory S by permitting up to k replications for an arbitrary motion of a query trajectory Q. When the motions of a data trajectory and a query trajectory are represented by rows and columns in the cumulative table respectively, our a k-warping distance algorithm finds a minimum distance starting from the first column of the first row within the last column of the last row by replicating an arbitrary motion of a query trajectory up to k times. In addition, since a motion of a trajectory is modeled as both angle property and distance property, our algorithm measures a similarity between a data trajectory S and a query trajectory Q by considering both properties.

```
int k-warping_distance(S, Q, k)
{
  Input:
     S[]: Data Trajectory;
     Q[]: Query Trajectory;
     k: the number of warping(replication);
  Output:
     kw_dist:minimum distance acquired using k-warping;

    kwTbl[MAXSIZE];   // k-warping table;

    for i=0 to |S|-1 do
      for j=0 to |Q|-1 do
          kwTbl[j+i*|Q|] = 999.0f;

    for i=0 to |S|-|Q| do {      // make k-warping table
      for n=0 to |Q|-1 do {
          y_p = i+n; x_p = n;
          kwTbl[x_p+(y_p*|Q|)] = $d_{df}$(S[y_p],Q[x_p]);
      } // end for n
      for j=0 to |Q|-1 do {
        for m=0 to k-1 do {
          for n=0 to |Q|-1 do {
            y_p = 1 + I + (j*k) + m + n;
            x_p = n;
            if((y_p>=|S|)||(y_p>x_p+(|S|-|Q|))) break;
            if(j == n)
              kwTbl[x_p+(y_p*|Q|)]=kwTbl[x_p+((y_p-1)*|Q|)] +
                       $d_{df}$(S[y_p],Q[x_p]);
            else
              kwTbl[x_p+(y_p*|Q|)]=$d_{df}$(S[y_p],Q[x_p])+
                             min(kwTbl[x_p+((y_p-1)*|Q|)],
                                 kwTbl[(x_p-1)+((y_p-1)*|Q|)]);
          } // end for n
        } // end for m
      } // end for j
    } // end for i

    kw_dist = 999.0f;   // initialize

    for i=0 to |S|-1 do {  // find the minimum k-warping dist.
      if(kw_dist > twTbl[(|Q|-1)+(i*|Q|)]) {
          kw_dist = twTbl[(|Q|-1)+(i*|Q|)];
          y = i; x = |Q|;
      }
    }
    return kw_dist;
}
```

Fig. 4. k-warping distance algorithm

Figure 5 depicts an example of our k-warping distance algorithm which can calculate similarity between trajectory S and Q when k is 2. We can permit up to 2(=k) times replications for an arbitrary motion of only query trajectory Q. In the above example, we can obtain the minimum distance value, that is, the maximum similarity value, between S and Q when q[1] of trajectory Q is mapped to each s[1], s[2], and s[3] of trajectory S, instead of the exact matching, namely, one-to-one mapping between trajectory S and Q. Therefore, it is shown that the approximate matching is superior to the exact mating for calculating the similarity between trajectories in spatio-temporal databases.

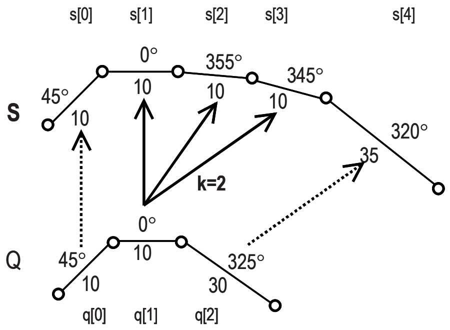


Fig. 5. Mapping of motions between S and Q when k=2

4.2 Similarity Measure for Single Trajectory

Based on our k-warping distance algorithm, we will define a similarity measure for a single trajectory. Since we measure a similarity between i-th motion in query trajectory Q and j-th motion in data trajectory S, we define a distance function between two motions, d_{df}(q[i], s[j]), as follows.

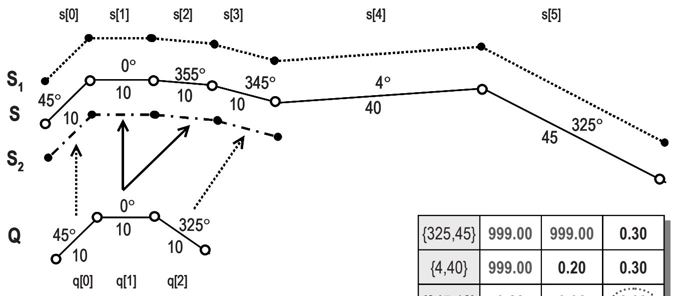
[Definition 14] A distance function, d_{df}(q[i], s[j]), to measure the similarity between the arbitrary motion s[i] of a data trajectory S and the arbitrary motion q[j] of a query trajectory Q is defined as follows.

$$d_{dis}(s[i,2], q[j,2]) = | s[i, 2] - q[j, 2] |$$
$$\text{if } | s[i, 1] - q[j, 1] | > 180 \text{ then } d_{ang}(s[i, 1], q[j, 1]) = (360 - | s[i, 1] - q[j, 1] |)$$
$$\text{else } d_{ang}(s[i, 1], q[j, 1]) = | s[i, 1] - q[j, 1] |$$

$$\mathbf{d_{df}(s[i], q[j])= (((d_{ang} / 180) * \alpha) + ((d_{dis}/100) * \beta))}$$

Here, d_{ang} is a distance function for the direction (angle) property for all the motions of a trajectory and d_{dis} is a distance function for the distance property. s[i, 1] and s[i, 2] are the direction and the distance value of the i-th motion in a trajectory S, respectively. α and β mean the weight of the direction and the distance, respectively, when $\alpha+\beta=1.0$.

For example, by using our k-warping distance algorithm, a similarity distance between a data trajectory S={(45,10), (0,10), (355,10), (345,10), (4,40), (325,45)} and a query trajectory Q={(45,10), (0,10), (325,10)} can be calculated in Figure 6. The value of the last column of the last row means the minimum distance 0.30 by permitting the infinitive replications of the query trajectory Q as shown in trajectory S1. In the case of k-warping distance, the motion of q[0] in the query trajectory Q corresponds to the s[0] in the data trajectory S, the motion of q[1] to the s[1], the motion of q[1] to the s[2], and the motion of q[2] to the s[3] respectively as shown in trajectory S2. Finally, we can find a path starting from the first column of the first row within the last column of the last row, thus obtaining the minimum distance by permitting up to k(=2) replications. We can summarize the differences of distance between each motion of the query and the data trajectory on the path, that is, |q[0]-s[0]|+|q[1]-s[1]|+|q[1]-s[2]|+|q[2]-s[3]|= 0.00 + 0.00 + 0.02 + 0.07 = 0.09. This is a minimum distance value between the two trajectories by using our k-warping distance algorithm. Thus, the similarity degree between S and Q is 91%(=1-0.09) while the similarity degree based on Shan's OCMR(infinite warping) is 70%(=1-0.30). In conclusion, our similarity measure scheme based on the k-warping distance algorithm provides a better result than Shan's OCMR.



	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
q[0]	0.00	0.17	0.02	0.04	0.24	0.41
q[1]	0.17	0.00	0.02	0.04	0.14	0.24
q[2]	0.31	0.13	0.11	0.07	0.24	0.10

{325,45}	999.00	999.00	0.30
{4,40}	999.00	0.20	0.30
{345,10}	0.23	0.06	0.09
{355,10}	0.19	0.02	0.11
{0,10}	0.17	0.00	999.00
{45,10}	0.00	999.00	999.00
S/Q	{45,10}	{0,10}	{325,10}

Fig. 6. Example of similarity measure between S and Q (k=2)

4.3 Similarity Measure for Multiple Trajectories

We first define the similarity measure for a relationship trajectory between two trajectories. For this, we makes use of topological relationships between multiple moving objects as well as moving direction and moving distance, by using our k-warping distance algorithm. Hence, we define a distance function using three-dimensional properties as follows.

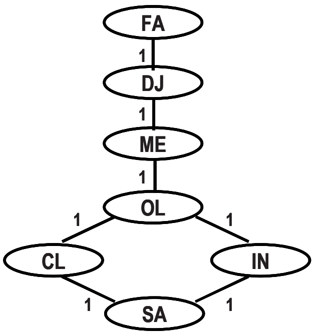
[Definition 15] A distance function, d_{df}(q[i], s[j]), to measure the similarity between the arbitrary motion s[i] of a data trajectory S and the arbitrary motion q[j] of a query trajectory Q is defined as follows.

$$d_{top}(s[i,3], q[j,3]) = (top\_dist(s[i, 3], q[j, 3]))^2$$
$$d_{dis}(s[i,2], q[j,2]) = | s[i, 2] - q[j, 2] |$$
$$\text{if } | s[i, 1] - q[j, 1] | > 180 \text{ then } d_{ang}(s[i, 1], q[j, 1]) = (360 - | s[i, 1] - q[j, 1] |)$$
$$\text{else } d_{ang}(s[i, 1], q[j, 1]) = | s[i, 1] - q[j, 1] |$$

$$\mathbf{d_{df}(s[i], q[j]) = (((d_{ang} / 180) * \alpha) + ((d_{dis}/100) * \beta) + ((d_{top}/25)*\gamma))}$$

Here d_{ang} is a distance function for the direction (angle) property for all the motions of a trajectory, d_{dis} is a distance function for the distance property and d_{top} is a distance function for the topology property. s[i, 1], s[i, 2] and s[i, 3] are the angle, the distance and the topology value of the i-th motion in a multiple trajectory S, respectively. α, β and γ mean the weight of the angle, the distance and the topology, respectively, when $\alpha+\beta+\gamma=1.0$. Also, top_dist(a, b) means the similarity distance for topological relations between a and b.

Figure 7 shows the similarity distance between a pair of seven topological relations operators. First, Figure 7(a) depicts a graph to describe their similarity for seven topological operators. We regard the distance value between adjacent nodes as '1'. Thus, Figure 7(b) shows the shortest distance between two nodes based on Figure 7(a). For example, the similarity distance between FA and DJ, FA and ME, FA and SA is '1', '2', and '5', respectively.



(a) Similarity graph for topological relation operators

top_dist (a,b)	FA	DJ	ME	OL	CL	IN	SA
FA	0	1	2	3	4	4	5
DJ	1	0	1	2	3	3	4
FA	2	1	0	1	2	2	3
DJ	3	2	1	0	1	1	2
FA	4	3	2	1	0	2	1
DJ	4	3	2	1	2	0	1
DJ	5	4	3	2	1	1	0

(b) Similarity distance

Fig. 7. Similarity distance between 7 topological relations operators

A list of multiple trajectories of at least two or more moving objects, MT(A_1, A_2, …, A_n), can be represented as a combination of single trajectory(ST) and relationship trajectories(RT).

[Definition 16] Let us suppose that i and j is the number of moving objects and stationary objects, respectively (n=i+j). The similarity for multiple trajectories of objects A_1, A_2, …, A_n, MT(A_1, A_2, …, A_n), is calculated as follows:

$$MT_{sim}(A_1, A_2,..., A_n) = \frac{\sum_{p=1}^{i} ST_{sim}(A_p) + \sum_{q=1}^{k} RT_{sim}(A_q, A_{q+1})}{i+k} \quad , k={}_nC_2 - {}_jC_2$$

Here $ST_{sim}(A_i)$ is the similarity value of single trajectory of object A_i. $RT_{sim}(A_k, A_{k+1})$ is the similarity value of relationship trajectories between objects A_k and A_{k+1} where k is the number of relationship trajectories between two moving objects as well as between a moving object and a stationary object.

For example, in Figure 2, we compute the similarity for multiple trajectories MT(C, M, B) of three objects: Car (C), Motorcycle (M), and Building (B). When we assume that $ST_{sim}(C)$, $ST_{sim}(M)$ is 0.8, 0.9 and $RT_{sim}(C,M)$, $RT_{sim}(C,B)$, $RT_{sim}(M,B)$ is 0.6, 0.75, 0.85, respectively, $MT_{sim}(C, M, B)$ is calculated as follows.

$$MT_{sim}(C,M,B) = \frac{((ST_{sim}(C)+ST_{sim}(M))+(RT_{sim}(C,M)+RT_{sim}(C,B)+RT_{sim}(M,B))}{i+k}$$

$$= \frac{((0.8+0.9)+(0.6+0.75+0.85))}{2+3} = \frac{3.9}{5} = 0.78$$

5 Performance Analysis

In order to verify the usefulness of our similarity measure scheme for both the single trajectory and the multiple trajectories, we do the performance analysis by using real soccer video data. Since the soccer video data has many trajectories of soccer balls, i.e., salient objects, we extract the trajectories of moving objects from the soccer ball. Table 1 describes the experimental data used for our performance analysis. Most of video data (formatted as MPEG file) used in our experiment include a shot of 'getting a goal'. We extract the trajectories of a soccer ball by manually tracing the ball in a ground field. For our experiment, we make forty query trajectories consisting of twenty in 'the right field' and twenty in 'the left field' from the half line of the ground field.

Table 1. Experimental data for performance analysis

Parameters	**Data Set**	
	Single Trajectory	**Multiple Trajectories**
Data domain	**Soccer Video Data**	
Salient moving object	**Soccer Ball**	**Soccer Ball and Player**
The number of data	**350**	**200**
The average motion number of data trajectory	**8.4**	**8.9**
The number of query	**40**	**10**
The average motion number of query trajectory	**3.8**	**3.1**

For our performance analysis, we implemented our similarity measure scheme based on our k-warping distance algorithm under Windows 2000 O.S with Pentium III-800 and 512 MB memory by using Microsoft Visual C++ compiler. We compare our similarity measure scheme based on the k-warping distance with the Li's (no-warping) and Shan's ones (infinite-warping) in terms of retrieval effectiveness, that is, average precision and recall measures [9]. Let RD (Relevant data in Database) be the number of video data relevant to a given query which are selected from the database, RQ

(Retrieved data by Query) be the total number of data retrieved by a given query, and RR (Relevant data that are Retrieved) be the number of relevant data retrieved by a given query. In order to obtain RD, we make a test panel which selects relevant data manually from the database. The test panel is composed of 20 graduate school students from the computer engineering department of Chonbuk National University, South Korea. The precision is defined as the proportion of retrieved data being relevant and the recall is defined as the proportion of relevant data being retrieved as follows.

$$\text{Precision} = \frac{RR}{RQ} \qquad \text{Recall} = \frac{RR}{RD}$$

For our performance comparison, we adopt the 11-point measure [9], which is most widely used for measuring the precision and recall. For a single trajectory, we consider the weight of angle (W_a) and the weight of distance (W_d) separately since we use both angle and distance for modeling the trajectory of moving objects. We also take into account the number of replications (k) since k is a very important parameter, depending on an application area. Here we do our experiment when k=0, 1, and 2 owing to the characteristics of the trajectory of the soccer ball in soccer video data. k=0 is exact matching and k=1 and 2 is approximate matching. We show from our experiment that there is no difference on retrieval effectiveness when k is greater than 2. Table 2 shows the retrieval effectiveness of our scheme, Li's scheme, and Shan's scheme. In case we do our performance analysis based on only the angle property (W_a=1.0 and W_d=0.0), it is shown that our scheme achieves about 10-15% higher precision than that of Li's and Shan's schemes while it holds about the same recall. In case we consider the weight of angle about two times greater than that of distance (W_a =0.7 and W_d=0.3), it is shown that our scheme achieves about 15-20% higher precision than that of Li's and Shan's schemes while it holds about the same recall. In case we consider the importance of the angle and that of the distance equally (W_a=0.5 and W_d=0.5), it is shown that our scheme is better than Li's and Shan's schemes in terms of both precision and recall measures.

Table 2. Performance result for single trajectory

		Avg. Precision			**Avg. Recall**		
	# of warping	**k = 0**	**k = 1**	**k = 2**	**k = 1**	**k = 1**	**k = 2**
W_a:W_d= 1.0:0.0	**Li's Scheme**	**0.25**			**0.45**		
	Shan's Scheme	**0.30**			**0.44**		
	Our Scheme	**0.34**	**0.38**	**0.40**	**0.51**	**0.48**	**0.47**
W_a:W_d= 0.7:0.3	**Li's Scheme**	**0.25**			**0.45**		
	Shan's Scheme	**0.30**			**0.44**		
	Our Scheme	**0.39**	**0.44**	**0.45**	**0.50**	**0.46**	**0.47**
W_a:W_d= 0.5:0.5	**Li's Scheme**	**0.25**			**0.45**		
	Shan's Scheme	**0.30**			**0.44**		
	Our Scheme	**0.33**	**0.34**	**0.38**	**0.51**	**0.50**	**0.51**

For multiple trajectories, we consider the weight of angle (W_a), the weight of distance (W_d) and the weight of topological relations (W_t) according to modeling the trajectory of multiple moving objects. When k is greater than 1, it is very difficult to obtain a relevant set for the multiple trajectories of a given query. Thus, we do our experiment for multiple trajectories when k=0 and 1. Table 3 depicts the performance results for multiple trajectories in our scheme, Li's scheme, and Shan's scheme. In case we consider the angle and the topological relation about two times more importantly than the distance (W_a=0.4, W_d=0.2, and W_t=0.4), it is shown that our scheme achieves about 20% higher precision than that of Li's and Shan's schemes while it holds about the same recall.

Table 3. Performance result for multiple trajectories

		Avg. Precision		Avg. Recall	
	# of warping	**k = 0**	**k = 1**	**k = 0**	**k=1**
W_a:W_d:W_t = 0.5:0.0:0.5	**Li's Scheme**	**0.37**		**0.49**	
	Shan's Scheme	**0.30**		**0.41**	
	Our Scheme	**0.39**	**0.52**	**0.48**	**0.52**
W_a:W_d:W_t = 0.4:0.2:0.4	**Li's Scheme**	**0.25**		**0.49**	
	Shan's Scheme	**0.30**		**0.41**	
	Our Scheme	**0.45**	**0.53**	**0.51**	**0.54**
W_a:W_d:W_t = 0.4:0.3:0.3	**Li's Scheme**	**0.25**		**0.49**	
	Shan's Scheme	**0.30**		**0.41**	
	Our Scheme	**0.41**	**0.46**	**0.51**	**0.52**

From our experiment, we finally show that our similarity measure scheme based on our k-warping distance algorithm achieves better performance on average precision than Li's and Shan's schemes while it holds about the same recall in the single trajectory and multiple trajectories. Particularly, in case of the single trajectory, the performance of our scheme is the best when the weight of angle is over two times than that of distance (W_a=0.7 and W_d=0.3). In case of the multiple trajectories, the performance of our scheme is the best when the weight of angle and topology is over two times than that of distance (W_a=0.4, W_d=0.2 and W_t=0.4). In terms of the number of replications (k), we show that the performance of our scheme when k=1 is better than the performance when k=0. Also, the performance when k=2 is better than the one when k=1. Thus, it is shown that the approximate matching is better on the performance of similar sub-trajectory retrieval than the exact matching, i.e., k=0.

6 Conclusions and Future Work

In this paper, we first proposed a new spatio-temporal representation scheme for modeling the trajectory of moving objects. Our spatio-temporal representation scheme effectively describes not only the single trajectory of a moving object but also the multiple trajectories of two or more moving objects. For measuring similarity between two trajectories, we proposed a new k-warping algorithm which enhances the existing time distance algorithm by permitting up to k replications for an arbitrary motion of a query trajectory. Based on our k-warping algorithm, we also presented a similarity measure scheme for both the single trajectory and the multiple trajectories in spatio-temporal databases. From our experiment, we showed that our similarity measure scheme based on the k-warping distance outperformed Li's (no-warping) and Shan's schemes (infinite-warping) in terms of precision and recall measures. In the result of the single trajectory, the performance of our scheme achieves about 15-20% performance improvement against Li's and Shan's scheme when the weight of angle is over two times greater than that of distance. In the result of the multiple trajectories, the performance of our scheme achieves about 20% performance improvement when the weights of angle and topological relation are over two times than that of distance. As future work, it is necessary to study on indexing methods to support good retrieval efficiency when the amount of trajectory data of moving objects is very large.

Acknowledgement. This work was supported in part by the Research Center of Industrial Technology, Engineering Research Institute at Chonbuk National University (CNU), South Korea.

References

[1] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider, "A Data Model and Data Structures for Moving Objects Databases", Proc. of ACM SIGMOD Conf, pp. 319–330, 2000.
[2] R. H. Guting, et al., "A Foundation for Representing and Querying Moving Objects", ACM Transaction on Database Systems, Vol. 25, No. 1, pp. 1–42, 2000.
[3] J. Z. Li, M. T. Ozsu, and D. Szafron, "Modeling Video Temporal Relationships in an Object Database Management System," in Proceedings of Multimedia Computing and Networking(MMCN97), pp. 80–91, 1997.
[4] J. Z. Li, M. T. Ozsu, and D. Szafron, "Modeling of Video Spatial Relationships in an Objectbase Management System," in Proceedings of International Workshop on Multimedia DBMS, pp. 124–133, 1996.
[5] M. K. Shan and S. Y. Lee, "Content-based Video Retrieval via Motion Trajectories," in Proceedings of SPIE Electronic Imaging and Multimedia System II, Vol. 3561, pp. 52–61, 1998.
[6] B. K. Yi, H. V. Lagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping," In Proc. Int'l. Conf. on Data Engineering, IEEE, pp. 201–208, 1998.
[7] S. H. Park, et al.,"Efficient Searches for Simialr Subsequence of Difference Lengths in Sequence Databases," In Proc. Int'l. Conf. on Data Engineering. IEEE, pp. 23–32, 2000.
[8] S. W. Kim, S. H. Park, and W. W. Chu, "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases," In Proc. Int'l. Conf. on Data Engineering. IEEE, pp. 607–614, 2001.
[9] H.S. Yoon, J. Soh, B.W. Min, and Y.K. Yang, "Soccer image sequences mosaicing using reverse affine transform," In Proc. of International Technical Conference on Circuits/Systems, Computers and Communications, pp. 877–880, 2000.

Distance Join Queries of Multiple Inputs in Spatial Databases

Antonio Corral[1]*, Yannis Manolopoulos[2], Yannis Theodoridis[3], and Michael Vassilakopoulos[4]

[1] Department of Languages and Computation, University of Almeria
04120 Almeria, Spain
acorral@ual.es

[2] Department of Informatics, Aristotle University
GR-54006 Thessaloniki, Greece
manolopo@csd.auth.gr

[3] Department of Informatics, University of Pireaus
GR-18534 Pireaus, Greece
ytheo@unipi.gr

[4] Department of Informatics, Technological Educational Institute of Thessaloniki
GR-54101 Thessaloniki, Greece
vasilako@it.teithe.gr

Abstract. Let a tuple of n objects obeying a query graph (QG) be called the n-tuple. The "$D_{distance}$-value" of this n-tuple is the value of a linear function of distances of the n objects that make up this n-tuple, according to the edges of the QG. This paper addresses the problem of finding the K n-tuples between n spatial datasets that have the smallest $D_{distance}$-values, the so-called K-Multi-Way Distance Join Query (K-MWDJQ), where each set is indexed by an R-tree-based structure. This query can be viewed as an extension of K-Closest-Pairs Query (K-CPQ) [4] for n inputs. In addition, a recursive non-incremental branch-and-bound algorithm following a Depth-First search for processing synchronously all inputs without producing any intermediate result is proposed. Enhanced pruning techniques are also applied to the n R-trees nodes in order to reduce the total response time of the query, and a global LRU buffer is used to reduce the number of disk accesses. Finally, an experimental study of the proposed algorithm using real spatial datasets is presented.

Keywords: Spatial databases, Distance join queries, R-trees, Performance study

1 Introduction

The term *spatial database* refers to a database that stores data from phenomena on, above or below the earth's surface, or in general, various kinds of multidimensional data of modern life [11]. In a computer system these data are represented

* Author supported by the Spanish project with ref. TIC 2002-03968.

by points, line segments, polygons, volumes and other kinds of 2D/3D geometric entities and are usually referred to as *spatial objects*.

Some typical spatial queries appearing in spatial databases are the following. (a) *point* (*range*) query; (b) *nearest neighbor* query (K-NNQ); (c) *pairwise* (*multi-way*) *join* query; and (d) *distance join* or *closest pair* query (K-CPQ). Moreover, processing of novel queries, like multi-way spatial join ones, has recently gained attention [9,10,13,14]. In the majority of those papers, a multi-way spatial join query is modeled by a query graph whose nodes represent spatial datasets and edges represent spatial predicates. One way to process this query is as a sequence of pairwise joins when all join spatial datasets are supported by spatial indexes or not (pipelined or build-and-match strategies, respectively). Another possible way, when all join spatial datasets are indexed (using e.g. R-trees), is to combine the filter and refinement steps in a synchronous tree traversal. Moreover, the research interest on distance-based queries involving two datasets (e.g. distance join queries) has increased in the last years, since they are appropriate for data analysis, decision making, etc. Given two datasets S1 and S2, the K-CPQ finds the K closest pairs of objects $<\mathrm{obj}1_i, \mathrm{obj}2_j>$, such that $\mathrm{obj}1_i \in S1$ and $\mathrm{obj}2_j \in S2$, with the K smallest distances between all possible pairs of objects that can be formed by choosing one object of S1 and one object of S2 [4,5,8,16]. If both S1 and S2 are indexed in R-trees, we can use the synchronous tree traversal with Depth-First or Best-First search for the query processing [4,5].

From the above, it is clear that the extension of distance join queries to n inputs with M predicates or constraints (like the multi-way joins) results in a novel query type, the so-called K-Multi-Way Distance Join Query (K-MWDJQ). To our knowledge, this query type has not been studied in the literature so far and this is the aim of this paper.

Definition: Given n non-empty spatial object datasets $S1, S2, \ldots, S_n$ and a query graph QG, the K-Multi-Way Distance Join Query retrieves the K distinct n-tuples of objects of these datasets with the K smallest $\mathrm{D}_{distance}$-values (i.e. the K $\mathrm{D}_{distance}$-smallest n-tuples).

The general environment for this kind of query can be represented by a *network*, where nodes correspond to spatial datasets and edges to binary metric relationships (distances), assigning positive real number to the edges. This framework is similar to the one defined in [10], where the graph is viewed as a constraint network: the nodes correspond to problem variables (datasets) and edges to binary spatial constraints (spatial predicates). Therefore, our network is a weighted directed graph, which directed edges correspond to binary metric relationships (e.g. distances) between pairs of spatial datasets (nodes) with specific weights (positive real numbers) and directions. We also assume that the weighted directed graph cannot be split into non-connected subgraphs. Of course, if we would have such a graph, it could be processed by solving all subgraphs and computing the appropriate combination.

K-Multi-Way Distance Join Queries are very useful in many applications using spatial data for decision making (e.g. in logistics) and other demanding data handling operations. For example, suppose we are given four spatial datasets consisting of the locations of *factories, warehouses, stores* and *customers*, connecting as Figure 1.a. A K-MWDJQ will find K different quadruplets (factory, warehouse, store, customer) that minimize a $\mathrm{D}_{distance}$ function (the K smallest $\mathrm{D}_{distance}$-values of the quadruplets are sorted in ascending order). Such a function would be, for example, the sum of distances between a factory and a warehouse, this warehouse and a store and this store and a customer. Such information could then be exploited by a transport agency or a fleet management system for different purposes (e.g. for minimizing transport cost). Moreover, the way to connect the spatial datasets could be more complex than a simple sequence. For example, in Figure 1.b, we can observe the case when the containers of products must be recycled from customers to factories through stores, and new distances must be considered for computing $\mathrm{D}_{distance}$. We have considered directed graphs instead of undirected graphs, because the former allow us the configuration of *itineraries* following a specific order; and the users can assign directions and weights to the arcs (directed edges) of the itineraries.

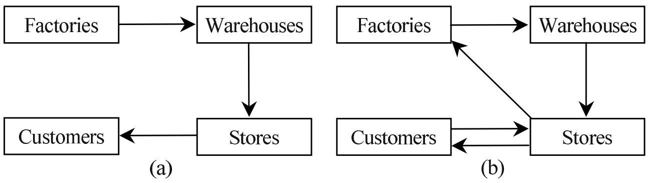


Fig. 1. Directed graphs for factories, warehouses, stores and customers.

The fundamental assumption in this paper is that the n spatial datasets are indexed by R-tree-based structures [7]. R-trees are hierarchical, height balanced multidimensional data structures in secondary storage, and they are used for the dynamic organization of a set of d-dimensional geometric objects represented by their Minimum Bounding d-dimensional hyper-Rectangles (MBRs). These MBRs are characterized by "min" and "max" points of hyper-rectangles with faces parallel to the coordinate axis. Using the MBR instead of the exact geometrical representation of the object, its representational complexity is reduced to two points where the most important features of the object (position and extension) are maintained. R-trees are considered an excellent choice for indexing various kinds of spatial data (points, line segments, polygons, etc.) and have already been adopted in commercial systems, such as Informix [3] and Oracle [12]. Moreover, we must highlight that, in case of non-point objects, an R-tree index can only organize objects' MBRs, together with the pointers to the place where their actual geometry has been stored. Under this framework, K-MWDJQ processing algorithms using R-trees produce only a set of n-tuples of MBRs (hence, candidate objects) in the filter step. For the refinement step, the exact geometry of the spatial objects has to be retrieved and exact distances have to be computed, according to the $D_{distance}$ function based on the query graph. The algorithm proposed in this paper addresses only the filter step.

The organization of the paper is as follows: In Section 2, we review the literature (K-closest-pairs queries and multi-way join queries) and motivate the research reported here. In Section 3, an expression for a linear distance function based on a given query graph, the definition of K-MWDJQ, an MBR-based distance function and a pruning heuristic are presented. In Section 4, enhanced pruning techniques and a recursive non-incremental branch-and-bound algorithm (called MPSR) for K-MWDJQ are proposed. Section 5 exhibits a detailed experimental study of the algorithm for K-MWDJQ, including the effect of the increase of K and n, in terms of performance (number of disk accesses and response time). Finally, in Section 6, conclusions on the contribution of this paper and related future research plans are summarized.

2 Related Work and Motivation

The K-Multi-Way Distance Join Query can be seen as a "combination" of K-Closest-Pairs query and Multi-way Spatial Join Query; therefore, we review these query types, focusing on the processing techniques of the query algorithms.

***K*-Closest-Pair (or Distance Join) Queries:** [4,5] presented recursive and iterative branch-and-bound algorithms for K-CPQ following a non-incremental approach, which computes the operation when K is known in advance and the K elements, belonging to the result, are reported all together at the end of the algorithm, i.e. the user can not have any result until the algorithm ends. The main advantage of this approach is that the pruning process during the execution of the algorithm is more effective even when K is large enough. On the other hand, the incremental approach for solving the distance join queries [8,16] computes the operation in the sense that the number of desired elements in the result is reported one-by-one in ascending order of distance (pipelined fashion), i.e. the user can have part of the final result before ending the algorithm. The incremental algorithms work in the sense that having obtained K elements in the result, if we want to find the $(K + 1)$-th, it is not necessary to restart the algorithm for $K + 1$, but just to perform an additional step.

Multi-way Spatial Join Queries: [9] proposed a pairwise join method that combines pairwise join algorithms in a processing tree where the leaves are input relations indexed by R-trees and the intermediate nodes are joins operator. [14] proposed a multi-way spatial join by applying systematic search algorithms that exploit R-trees to efficiently guide search, without building temporary indexes or materializing intermediate results. On the other hand, [13] proposed a multi-way R-tree join as a generalization of the original R-tree join [2], taking into account its optimization techniques (the search space restriction and the plane-sweep method). In addition, a recent work [10] reviews pairwise spatial join algorithms and shows how they can be combined for multiple inputs, explores the applications of synchronous tree traversal for processing synchronously all inputs without producing intermediate results; and then, integrates the two approaches (synchronous tree traversal and pairwise algorithms) in an *engine* using dynamic programming to determine the optimal execution plan.

All the previous efforts have mainly been focused over multi-way spatial join queries, using a sequence of pairwise join algorithms or synchronous tree traversal over R-tree structures on the filter step. Thus, the main objective of this paper is to investigate the behavior of recursive branch-and-bound algorithms in a non-incremental manner for K-MWDJQ as a generalization of K-CPQ between n spatial datasets indexed by R-trees, without producing any intermediate result. To do this, we extend the distance metrics and the pruning heuristic based on the query graph for solving this kind of distance-based query. In addition, we apply techniques for improving the performance with respect to the I/O activity (buffering) and response time (distance-based plane-sweep) in our experiments over real spatial datasets of different nature (points and line segments).

3 K-Multi-Way Distance Join Queries Using R-Trees

Let us recall the assumptions we make: (a) n spatial datasets are involved, each supported by an R-tree structure; (b) M ($M \geq n - 1$) spatial predicates (metric relationships) between pairs of objects are defined; and (c) a query graph declares the spatial constraints that have to be fulfilled. In the following, we state more formally the details of the problem.

3.1 The Query Graph and the $\mathbf{D}_{distance}$ Function

Query Graph (QG). A query graph QG = (S, E) is a weighted directed graph which consists of a finite non-empty set of nodes $S = \{s_1, s_2, \ldots, s_n\}$ and a finite set of directed edges $E = \{e_{i,j} = (s_i \to s_j) : s_i, s_j \in S\}$. Each directed edge $e_{i,j}$ connects an ordered pair of nodes $(s_i \to s_j)$, where s_i and s_j are called start and end nodes of the directed edge, respectively. Associated with each directed edge $e_{i,j}$, there exists a weight $w_{i,j}$, which is a positive real number ($w_{i,j} \in \Re^+$). We assume that the reader is familiar with the related concepts of weighted directed graphs (path, circuit, cycle, self-loop, etc.).

Different configurations of QG depending on the required results by the users are possible. Examples include *sequential* or *"chain"* queries (Figure 1.a), where QG is a *acyclic* weighted directed graph among all datasets, obeying the constraints of a *simple directed path* and it does not contain any directed circuit. Queries with cycles (Figure 1.b) correspond to a QG, with at least one *simple directed circuit* among the nodes existing there (i.e. ordered sequences of nodes with simple directed circuits). Based on this query graph definition, we can define the $\mathrm{D}_{distance}$ function as follows:

$\mathbf{D}_{distance}$ Function. Given n non-empty spatial object datasets $S1, S2, \ldots,$ Sn, organized according to a query graph QG and an n-tuple t of spatial objects of these datasets, $\mathrm{D}_{distance}(t)$ is a linear function of distance values of the pairs of spatial objects belonging in the n-tuple t that result from the directed edges of QG. More formally, we can define $\mathrm{D}_{distance}(t)$ as follows:

$$D_{distance}(t) = \sum_{e_{i,j} \in E_{QG}} w_{i,j}\, distance(obj_i, obj_j)$$

where $t = (obj_1, obj_2, \ldots, obj_n) \in S1 \times S2 \times \ldots \times Sn$, the datasets of the objects of the ordered pair (obj_i, obj_j) are connected in QG by the directed edge $e_{i,j}$, $w_{i,j} \in \Re^+$ is the weight of $e_{i,j}$ and *distance* may represent any Minkowski distance norm (Euclidean, Manhattan, etc.) between pairs of spatial objects.

3.2 Definition of the *K*-Multi-way Distance Join Query

We define the K-Multi-Way Distance Join Query in the spatial database environment as follows:

***K*-Multi-way Distance Join Query.** Let n non-empty spatial object datasets $S1, S2, \ldots, Sn$, organized according to a query graph QG, where a $\mathrm{D}_{distance}$ function is defined. Assume that $obj \subseteq E^d$ (E^d denotes the d-dimensional Euclidean space), for each object obj of any of the above datasets. The result of the K-Multi-Way Distance Join Query, K-MWDJQ($S1, S2, \ldots,$ Sn, QG, K), is a set of ordered sequences of K $(1 \leq K \leq | S1 | \cdot | S2 | \cdot \ldots \cdot | Sn |)$ different n-tuples of spatial objects of $S1 \times S2 \times \ldots \times Sn$, with the K smallest $\mathrm{D}_{distance}$-values between all possible n-tuples of spatial objects that can be formed by choosing one spatial object from each spatial dataset (i.e. the K $\mathrm{D}_{distance}$-smallest n-tuples):

$$\begin{gathered} K\text{-MWDJQ}(S1, S2, \ldots, Sn, QG, K) = \\ \{(t_1, t_2, \ldots, t_K) : t_i \in S1 \times S2 \times \ldots \times Sn \text{ and } t_i \neq t_j \ \forall i \neq j, 1 \leq i, j \leq K \text{ and} \\ \forall t \in S1 \times S2 \times \ldots \times Sn - \{(t_1, t_2, \ldots, t_K)\}, D_{distance}(t) \geq D_{distance}(t_K) \geq \\ D_{distance}(t_{(K-1)}) \geq \ldots \geq D_{distance}(t_2) \geq D_{distance}(t_1)\} \end{gathered}$$

In other words, the K $\mathrm{D}_{distance}$-smallest n-tuples from the n spatial object datasets obeying the query graph QG are the K n-tuples that have the K smallest $\mathrm{D}_{distance}$-values between all possible n-tuples of spatial objects that can be formed by choosing one spatial object of $S1$, one spatial object of $S2, \ldots$, and one spatial object of Sn. Of course, K must be smaller than or equal to $| S1 | \cdot | S2 | \cdot \ldots \cdot | Sn |$ ($| Si |$ is the cardinality of the dataset Si) i.e. the number of possible n-tuples that can be formed from $S1, S2, \ldots, Sn$. Note that, due to ties of $\mathrm{D}_{distance}$-values, the result of the K-Multi-Way Distance Join Query may not be unique for a specific K and a set of n spatial datasets $S1, S2, \ldots, Sn$. The aim of the presented algorithm is to find one of the possible instances, although it would be straightforward to obtain all of them.

3.3 MBR-Based Distance Function and Pruning Heuristic

The following distance functions between MBRs in E^d have been proposed for the K-CPQ [4,5] as bounds for the non-incremental branch-and-bound algorithms: MINMINDIST (it determines the minimum distance between two MBRs, and it is a generalization of the function that calculates the minimum distance between points and MBRs), MINMAXDIST and MAXMAXDIST.

In the following the definition of the new metric, called $D_{MINMINDIST}$, between n MBRs that depends on the query graph and is based on MINMINDIST distance function between two MBRs in E^d (i.e. $D_{MINMINDIST}$ can be viewed as an instance of $\mathrm{D}_{distance}$ for MINMINDIST function) is presented.

MINMINDIST Function. Let $M(A,B)$ represent an MBR in E^d, where $A=(a_1,a_2,\ldots,a_d)$ and $B=(b_1,b_2,\ldots,b_d)$, such that $a_i \le b_i$, $1 \le i \le d$, be the endpoints of one of its major diagonals. Given two MBRs $M_1(A,B)$ and $M_2(C,D)$ in E^d, MINMINDIST$(M_1(A,B), M_2(C,D))$ is defined as:

$$\mathrm{MINMINDIST}(M_1,M_2)=\sqrt{\sum_{i=1}^{d} {y_i}^2}, \quad y_i=\begin{cases} c_i-b_i, & \text{if } c_i>b_i \\ a_i-d_i, & \text{if } a_i>d_i \\ 0, & \text{otherwise} \end{cases}$$

$D_{\mathbf{MINMINDIST}}$ Function. Let $M(A,B)$ represent an MBR in E^d, where $A=(a_1,a_2,\ldots,a_d)$ and $B=(b_1,b_2,\ldots,b_d)$, such that $a_i \le b_i$, $1 \le i \le d$, are the endpoints of one of its major diagonals. R_{S_i} is the R-tree associated to the dataset S_i and QG is a query graph obeyed by the n R-trees $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$. Given an n-tuple t of MBRs stored in n R-trees (i.e. t is a tuple of n MBRs from $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$) $D_{MINMINDIST}(t)$ is a linear function of MINMINDIST distance function values of the pairs of t that result from the edges of QG. More formally, we can define $D_{MINMINDIST}(\mathrm{t})$ as follows:

$$D_{MINMINDIST}(t)=\sum_{e_{i,j}\in E_{QG}} w_{i,j}\,MINMINDIST(M_i,M_j)$$

where $t=(M_1,M_2,\ldots,M_n)$ with M_i an MBR of the R-tree R_{S_i} $(1 \le i \le n)$, the R-trees of the MBRs of the ordered pair (M_i,M_j) are connected by the directed edge $e_{i,j}$ in QG and $w_{i,j} \in \Re^+$ is the weight of $e_{i,j}$. In other words, $D_{MINMINDIST}$ represents our $\mathrm{D}_{distance}$ function based on MINMINDIST metric for each possible pair of MBRs that belongs in the n-tuple t and satisfies QG.

$D_{MINMINDIST}$ expresses the minimum possible distance of any n-tuple containing n MBRs. For example, in Figure 2, six MBRs (a *6*-tuple of MBRs, $t=(M_{11},M_{23},M_{32},M_{44},M_{52},M_{65})$) and their MINMINDIST distances are depicted for a sequential query (QG $=(S_1 \to S_2 \to S_3 \to S_4 \to S_5 \to S_6)$). $D_{MINMINDIST}$ represents the sum of their MINMINDIST distance values.

We can extend the same properties of MINMINDIST metric between two MBRs to the $D_{MINMINDIST}$ for an n-tuple of MBRs. The most basic properties of $D_{MINMINDIST}$ are the following:

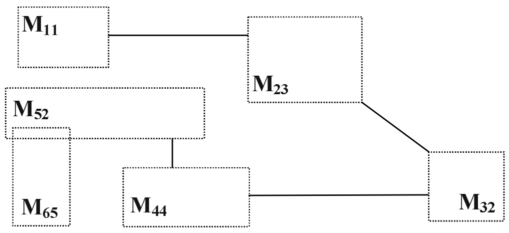


Fig. 2. Example of $D_{MINMINDIST}$ for a sequential query.

(a) Given an n-tuple t of MBRs, the value of $D_{MINMINDIST}$, for a given dimension i, $1 \leq i \leq d$, is always smaller than or equal to the total computation of $D_{MINMINDIST}$: $D_{MINMINDIST}(t, i) \leq D_{MINMINDIST}(t), 1 \leq i \leq d$
(b) Lower-bounding property. For each n-tuple t of spatial objects, enclosed by a n-tuple of MBRs t', it holds that: $D_{MINMINDIST}(t') \leq D_{distance}(t)$
(c) $D_{MINMINDIST}$, like MINMINDIST, is monotonically non-decreasing with the R-tree heights. This means that, for a given n-tuple t of MBRs enclosed by another n-tuple of MBRs t' (where each MBR of t' covers its respective MBR in t), it holds that: $D_{MINMINDIST}(t') \leq D_{MINMINDIST}(t)$

In [5], a pruning heuristic (based on MINMINDIST) was presented in order to minimize the pruning distance (distance value of the K-th closest pair that has been found so far) during the processing of branch-and-bound algorithms for K-CPQ. It declared that, *if MINMINDIST*$(M_1, M_2) > z$*, then the pair of MBRs* (M_1, M_2) *can be discarded*, where z can be obtained from the distance of the K-th closest pair of spatial object found so far. We can extend this pruning heuristic for our new $D_{MINMINDIST}$ function as follows: *if* $D_{\text{MINMINDIST}}(t) > z$*, then the n-tuple of MBRs t can be pruned, where z is the* $D_{distance}$*-value of the K-th n-tuple of spatial objects discovered so far.*

4 An Algorithm for *K*-Multi-way Distance Join Queries

In this section, based on $D_{MINMINDIST}$ function and the pruning heuristic, we are going to propose a recursive non-incremental algorithm for solving the K-Multi-Way Distance Join Query, processing all inputs (n R-trees, indexing n spatial datasets) without producing any intermediate result. This recursive algorithm follows a Depth-First search between n spatial objects indexed in n R-trees. Moreover, enhanced pruning techniques are used in the pruning process for avoiding considering all possible n-tuples of MBRs from n R-tree nodes.

4.1 Enhancing the Pruning Process

An improvement over branch-and-bound algorithms consists in exploiting the spatial structure of the indexes using the plane-sweep technique [15]. We extend the distance-based plane-sweep technique proposed in [5] for K-CPQ in order to restrict all possible combinations of n-tuples of MBRs from n R-tree nodes in a similar way as in the processing of multi-way join query presented in [10].

Plane-sweep is a common technique for computing intersections [15]. The basic idea is to move a line, called *sweep-line*, perpendicular to one of the dimensions, e.g. X-axis, from left to right. We apply this technique for restricting all possible combinations of n-tuples of MBRs from n R-tree nodes stored in n R-trees. If we do not use this technique, then we must create a list with all possible combinations of n-tuples from n R-tree nodes and process it.

The distance-based plane-sweep technique starts by sorting the entries of the n current R-tree nodes $N_i(1 \leq i \leq n)$ from the n R-trees, based on the coordinates of one of the corners of their MBRs (e.g. lower left corner) in increasing or decreasing order (according to the choice of the sweeping direction and the sweeping dimension, based on the sweeping axis criteria [16]). Suppose that this order is increasing and that Sweeping_Dimension = 0, or X-axis. Then, a set of n pointers (one for each R-tree node) is maintained, initially pointing to the first entry of each X-sorted R-tree node. Among all these entries, let $E_{ix} \in N_i$ ($1 \leq x \leq C_{N_i}$, where C_{N_i} is the capacity of the R-tree node N_i) be the one with the smallest X-value of lower left corner of MBR. We fix the current pivot $\mathbf{P} = E_{ix}$. The MBR of the pivot $\mathbf{P}$ must be paired up with all the MBRs of the entries of the other $n-1$ R-tree nodes $N_j(1 \leq j \leq n$ and $j \neq i)$ from left to right that satisfy MINMINDIST($\mathbf{P}$.MBR, E_{jy}.MBR, Sweeping_Dimension) $\leq z$, where E_{jy} ($1 \leq y \leq C_{N_j}$) is an entry of the R-tree node N_j and z is the $\mathrm{D}_{distance}$-value of the K-th n-tuple of spatial objects found so far. A set of n-tuples of MBRs, ENTRIES = $\{t_1, t_2, \ldots\}$ (empty at the beginning), is obtained. After all these n-tuples of MBRs been processed, the pointer currently pointing E_{ix} is advanced to the next entry of N_i (according to X-order), $\mathbf{P}$ is updated with the entry with the next smallest value of lower left corner of MBR pointed by one of the n pointers, and the process is repeated.

Notice that we apply MINMINDIST(M_{ix}, M_{jy}, Sweeping_Dimension) because the distance over one dimension between a pair of MBRs is always smaller than or equal to their MINMINDIST(M_{ix}, M_{jy}) (a direct extension of the property of MINMINDIST distance function [5]). Moreover, the searching is restricted only to the closest MBRs (belonging to the remainder $n-1$ R-tree nodes) from the pivot $\mathbf{P}$ according to the z value, and no duplicated n-tuples are obtained because the rectangles are always checked over sorted R-tree nodes. The application of this technique can be viewed as a *sliding window* on the sweeping dimension with a width equal to z plus the length of the MBR of the pivot $\mathbf{P}$ on the sweeping dimension, where we only choose all possible n-tuples of MBRs that can be formed using the MBR of the pivot $\mathbf{P}$ and the other MBRs from the remainder $n-1$ R-tree nodes that fall into the current sliding window.

For example, Figure 3 illustrates three sets of MBRs of ($n =$) 3 R-tree nodes: $\{M_{P1}$, M_{P2}, M_{P3}, M_{P4}, M_{P5}, $M_{P6}\}$, $\{M_{Q1}$, M_{Q2}, M_{Q3}, M_{Q4}, M_{Q5}, M_{Q6}, $M_{Q7}\}$, and $\{M_{R1}$, M_{R2}, M_{R3}, M_{R4}, M_{R5}, $M_{R6}\}$. Without applying this technique we should generate 6*7*6 = 252 triples of MBRs and process them. If we apply the previous method over the X-axis (sweeping dimension), this number of possible triples will be considerably reduced. First of all, we fix the MBR of the pivot $\mathbf{P} = M_{P1}$ and it must be paired up with $\{M_{Q1}, M_{Q2}, M_{Q3}, M_{Q4}\}$ and $\{M_{R1}, M_{R2}, M_{R3}\}$ because all triples that can be formed from them have MINMINDIST(M_{P1}, M_{Ry}, Sweeping_Dimension) $\leq z$ and the other MBRs can be discarded: $\{M_{Q5}, M_{Q6}, M_{Q7}\}$ and $\{M_{R4}, M_{R5}, M_{R6}\}$. In this case, we will obtain a set of 12 triples of MBRs: $\{(M_{P1}, M_{Q1}, M_{R1}), (M_{P1}, M_{Q1}, M_{R2}), (M_{P1}, M_{Q1}, M_{R3}), (M_{P1}, M_{Q2}, M_{R1}), \ldots, (M_{P1}, M_{Q4}, M_{R3})\}$. When processing is finished with $\mathbf{P} = M_{P1}$, the algorithm must establish the pivot $\mathbf{P} = M_{Q1}$ that is the next smallest value of lower left corner and the process is repeated. At the end, the number of triples of MBRs is 193 = |ENTRIES| (we save 59 *3*-tuples).

After obtaining a reduced set of candidate n-tuples of MBRs from n R-tree nodes (ENTRIES), applying the distance-based plane-sweep technique, we can consider the $D_{MINMINDIST}$ function based on the query graph (QG) over the Sweeping_Dimension as another improvement of the pruning process. Thus, we will only choose for processing those n-tuples of MBRs for which it holds that $D_{MINMINDIST}(t$, Sweeping_Dimension$) \leq z$. This is called $D_{MINMINDIST}$-Sweeping_Dimension filter (i.e. apply the pruning heuristic over the Sweeping_Dimension, preserving the order of entries in this dimension). In the previous example of Figure 3, we can reduce the number of *3*-tuples of MBRs (ENTRIES), depending on the organization of the query graph. If it is a sequential query ($R_P \rightarrow R_Q \rightarrow R_R$) and $\mathbf{P} = M_{P1}$, then the *3*-tuples of MBRs $\{(M_{P1}, M_{Q4}, M_{R1}), (M_{P1}, M_{Q4}, M_{R2})\}$ can be discarded. At the end of the processing of this second filter |ENTRIES| = 164 (we save 29 *3*-tuples). On the other hand, if the query graph is a cycle ($R_P \rightarrow R_Q \rightarrow R_R \rightarrow R_P$) and $\mathbf{P} = M_{P1}$, then the *3*-tuples of MBRs $\{(M_{P1}, M_{Q2}, M_{R3}), (M_{P1}, M_{Q3}, M_{R2}), (M_{P1}, M_{Q3}, M_{R3}), (M_{P1}, M_{Q4}, M_{R1}), (M_{P1}, M_{Q4}, M_{R2}), (M_{P1}, M_{Q4}, M_{R3})\}$ can be discarded, considering only a set of 6 *3*-tuples of MBRs. At the end of the processing of this second filter |ENTRIES| = 107 (we save 86 *3*-tuples). In summary, the pruning process over n R-tree nodes consists of two consecutive filters:

(1) Apply the distance-based plane-sweep technique: select all possible n-tuples of MBRs that can be formed using an MBR as pivot and the others MBRs from the remainder $n-1$ R-tree nodes that fall into a *sliding window* with width equal to z plus the length of the pivot MBR on the Sweeping_Dimension (ENTRIES), since MINMINDIST(M_{ix}, M_{jy}, Sweeping_Dimension) $\leq$ MINMINDIST(M_{ix}, M_{jy}).
(2) Apply the $D_{MINMINDIST}$-Sweeping_Dimension filter: consider from ENTRIES, only those n-tuples of MBRs that satisfy $D_{MINMINDIST}(t$, Sweeping_Dimension$) \leq z$, since $D_{MINMINDIST}(t$, Sweeping_Dimension$) \leq D_{MINMINDIST}(t)$.

4.2 A Recursive Branch-and-Bound Algorithm for *K*-MWDJQ

The recursive non-incremental branch-and-bound algorithm follows a Depth-First searching strategy making use of recursion and the previous pruning heuristic based on the $D_{MINMINDIST}$ function. In addition, we employ the distance-based plane-sweep technique and $D_{MINMINDIST}$-Sweeping_Dimension filter for

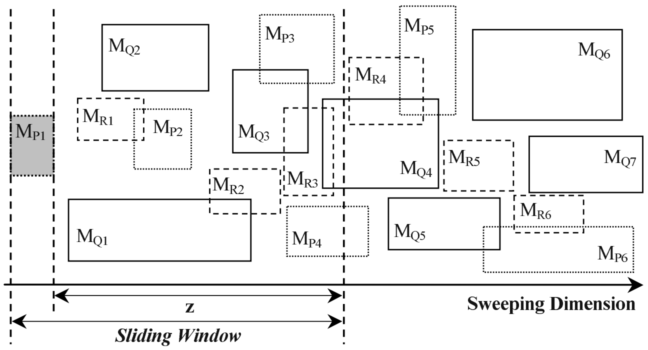


Fig. 3. Using the plane-sweep technique over the MBRs from three R-tree nodes.

obtaining a reduced set of candidate n-tuples of entries from n R-tree nodes (ENTRIES). Then, it iterates over the ENTRIES set and propagates downwards only for the n-tuples of entries with $D_{MINMINDIST}$-value smaller than or equal to z ($\mathrm{D}_{distance}$-value of the K-th n-tuple of spatial objects found so far). Also, we need an additional data structure, organized as a maximum binary heap (called K-heap) that holds n-tuples of spatial objects according to their $\mathrm{D}_{distance}$-values and stores the K $\mathrm{D}_{distance}$-smallest n-tuples and helps us to update z (pruning distance). The **MPSR** algorithm (extension of the PSR algorithm [5] for K-Multi-Way Distance Join Query) for n R-trees storing spatial objects (points or line-segments) on the leaf nodes, with the same height can be described by the following steps.

MPSR1. Start from the roots of the n R-trees and set z to ∞.

MPSR2. If you access to a set of n internal nodes, apply the distance-based plane-sweep technique and the $D_{MINMINDIST}$-Sweeping_Dimension filter in order to obtain the set of n-tuples of candidate MBRs, ENTRIES. Propagate downwards recursively only for those n-tuples of MBRs from ENTRIES that have $D_{MINMINDIST}$-value $\leq z$.

MPSR3. If you access a set of n leaf nodes, apply the distance-based plane-sweep technique and the $D_{MINMINDIST}$-Sweeping_Dimension filter to obtain the set of candidate n-tuples of entries, ENTRIES. Then calculate the $\mathrm{D}_{distance}$-value of each n-tuple of spatial objects stored in ENTRIES. If this distance is smaller than or equal to z, remove the n-tuple of spatial objects in the root of K-heap and insert the new one, updating z and K-heap.

In general, the algorithm synchronously processes the n R-tree indexes of all spatial datasets involved in the query (following a Depth-First traversal pattern), using the combinations of R-tree nodes reported by the application of the distance-based plane-sweep technique and $D_{MINMINDIST}$-Sweeping_Dimension filter that satisfy the query graph and pruning the n-tuples with $D_{MINMINDIST}$-value (n internal nodes) or $\mathrm{D}_{distance}$-value (n leaf nodes) larger than z.

The advantage of the algorithm that synchronously traverses, with a Depth-First search strategy, all R-trees is that it transforms the problem in smaller local subprolems at each tree level and it does not produce any intermediate result. The downward propagation in step **MPSR2** is done in the order produced by the distance-based plane-sweep technique; and this order is quite good, since it leads to very accurate results quickly. In addition, the algorithm consumes an amount of space that is only a linear function of the heights of the trees and n (number of inputs), and its implementation is relatively easy, because we can use of the recursion. A disadvantage of this algorithm (Depth-First search) is that it tends to consume time to exit, once it deviates from the branches where no optimal solutions of the initial problem are located and the recursion is more expensive with the increase of n.

5 Experimental Results

This section provides the results of an experimentation study aiming at measuring and evaluating the efficiency of the MPSR algorithm. In our experiments, we have used the R*-tree [1] as the underlying disk-resident access method and a global LRU buffer over the n R*-trees with 512 pages. R*-trees nodes, disk pages and buffer pages will have the same size. If the R*-trees have different heights, we will use the fix-at-leaves technique [4]. All experiments were run on an Intel/Linux workstation at 450 MHz with 256 Mbytes RAM and several Gbytes of secondary storage.

In order to evaluate the behavior of the K-MWDJQ algorithm, we have used four real spatial datasets of North America in the same workspace from [6], representing (a) populated places (NApp) consisting of 24,493 2d-points, (b) cultural landmarks (NAcl) consisting of 9,203 2d-points, (c) roads (NArd) consisting of 569,120 line-segments, and (d) railroads (NArr) consisting of 191,637 line-segments. Besides, we have generated (e) a 'pseudo-real' dataset from the 'populated places', simulating archeological places (NAap) of North America and consisting of 61,012 2d-points. With all these datasets, we have designed the following configurations for our experiments, where SQ represents *sequential query* and CY means *query with cycles* in the query graph (the weights of the directed edges: $w_{i,j} = 1.0$).

- $n = 2$: K-MWDJQ(NApp, NAcl, QG, K): QG = (NApp $\rightarrow$ NAcl).
- $n = 3$: K-MWDJQ(NApp, NArd, NAcl, QG, K): QG$_{SQ}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NAcl); QG$_{CY}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NAcl $\rightarrow$ NApp).
- $n = 4$: K-MWDJQ(NApp, NArd, NArr, NAcl, QG, K): QG$_{SQ}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NArr $\rightarrow$ NAcl); QG$_{CY}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NArr $\rightarrow$ NAcl $\rightarrow$ NArr $\rightarrow$ NApp).
- $n = 5$: K-MWDJQ(NApp, NArd, NAap, NArr, NAcl, QG, K): QG$_{SQ}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NAap $\rightarrow$ NArr $\rightarrow$ NAcl); QG$_{CY}$ = (NApp $\rightarrow$ NArd $\rightarrow$ NAap $\rightarrow$ NArr $\rightarrow$ NAcl $\rightarrow$ NArr $\rightarrow$ NApp).

We have measured the performance of our algorithm based on the following two metrics: (1) number of Disk Accesses (DA), which represents the number

Table 1. Comparison of the MPSR algorithm, varying the R*-tree node size.

	1/2 Kbyte		**1 Kbyte**		**2 Kbytes**		**4 Kbytes**		**8 Kbytes**	
n	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*
2	2029	0.35	996	0.41	492	0.43	237	0.46	122	0.51
3	32158	16.95	17884	19.39	9436	27.99	4477	34.28	2250	49.75
4	39720	423.38	24551	932.36	13292	2765.1	6384	16603.1	3041	17723.2

of R*-tree nodes fetched from disk, and may not exactly correspond to actual disk I/O, since R*-tree nodes can be found in system buffers, and (2) Response Time (RT), which is reported in seconds and represents the overall CPU time consumed, as well as the total I/O performed by the algorithm (i.e. the total query time). Moreover, due to the different nature of the spatial objects (points and line-segments) involved in the query, we have implemented the minimum distances between points and line-segments.

The first experiment studies the page size for the K-MWDJQ algorithm (MPSR), since the smaller the R*-tree node size is, the smaller the number of n-tuples of R*-tree items have to be considered in the algorithm. We have adopted the following query configurations for MPSR: n = 2, 3 and 4; QG$_{SQ}$ (sequential query graphs) and $K = 100$. Table 1 compares the performance measurements for different R*-tree node sizes, where Cmax is the maximum R*-tree node capacity: 1/2 Kbyte (Cmax = 25), 1 Kbyte (Cmax = 50), 2 Kbytes (Cmax = 102), 4 Kbytes (Cmax = 204) and 8 Kbytes (Cmax = 409). We use as minimum R*-tree node capacity Cmin = $\lfloor$Cmax*0.4$\rfloor$, according to [1], for obtaining the best query performance.

We can observe from the previous table that the smaller the R*-tree node size is, the faster the MPSR algorithm is, although it obviously needs more disk accesses (using a global LRU buffer, which minimizes the extra I/O cost). As expected, there is a balance between I/O activity (DA) and CPU cost (RT). Since deriving the optimum page size is an unmanageable task due to the number of parameters, we rather focus on the algorithmic issues and not on the previous question. On the other hand, hardware developments are rapid and manufacturers provide disks with larger page sizes year-after-year. Thus, we provide results for the case of page size (R*-tree node size) equal to 1 Kbyte (resulting in the following values of the branching factors for the R*-trees: Cmax = 50 and Cmin = 20, and the reader can extrapolate the method performance for other page sizes. For example, if we compare the size 1 Kbyte and 4 Kbytes for $n = 4$, 1 Kbyte becomes faster than 4 Kbytes by a factor of 17.8, although the increase of disk accesses is only a factor 3.8 using a global LRU buffer with 512 pages.

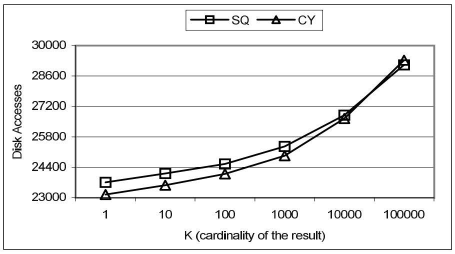
The second experiment studies the behavior of MPSR algorithm as a function of the number of the spatial datasets involved in the K-MWDJQ. In particular, we use $n = 2$, 3, 4 and 5, $K = 100$, SQ and CY (configurations of the query graph) as the algorithmic parameters for the query. Table 2 shows the performance measurements (DA and RT) of the experiment. We can observe that the increase of the response time is almost exponential with respect to the number of inputs,

Table 2. Comparison of the MPSR algorithm, as a function of the number of inputs.

	$n = 2$		$n = 3$		$n = 4$		$n = 5$	
	DA	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*
SQ	996	0.45	17884	19.43	24551	932.36	42124	93942.51
CY			17773	26.47	24088	1169.20	38947	120550.21

whereas the increase of the number of disk accesses is almost linear. This is due to the fact that the number of distance computations depends on the number of considered items in the combination of n R-tree nodes; and it is an exponential function of the R-tree structures (fan-outs: Cmin and Cmax, heights, etc.) and n (Cmin^n). For example, if we compare $n = 4$ and $n = 5$ with QG$_{SQ}$, the increases of DA and RT are by a factor of 1.7 and 100.7, respectively. Therefore, we can conclude that the response time is more affected than the disk accesses with the increase of the number of inputs in this kind of distance-based query.

The last experiment studies the performance of the MPSR algorithm with respect to the increase of K (number of n-tuples in the result) values, varying from 1 to 100000. Figure 4 illustrates the performance measurements for the following configuration: $n = 4$ (for $n = 3$, the trends were similar), SQ and CY. We can notice from the left chart of the figure that the I/O activity of the algorithm gets higher as K increases and both query graph configurations have similar I/O trends. Moreover, the deterioration is not smooth, although the increase of DA from $K = 1$ to $K = 100000$ is only around a 20%. In the right diagram, we can see that the larger the K values are, the slower the MPSR algorithm becomes, mainly for large K values. For instance, when $K = 1$ and $K = 10000$, the algorithm becomes slower by a factor of 6, and from $K = 1$ to $K = 100000$ the algorithm is 23 times slower for SQ and 27 for CY. From these results, we must highlight the huge response time necessary to report the exact result for large K values and the very small number of required disk accesses.



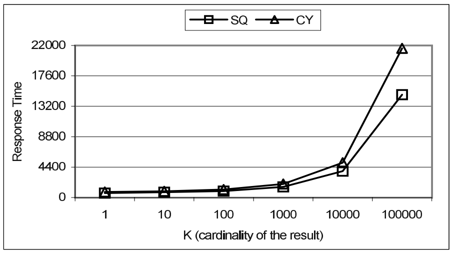


Fig. 4. Performance comparison of the MPSR algorithm for K-MWDJQ varying K for disk accesses (left) and response time (right).

6 Conclusions and Future Work

In this paper, we have presented the problem of finding the K n-tuples between n spatial datasets that have the smallest $\mathrm{D}_{distance}$-values, K-Multi-Way Distance Join Query (K-MWDJQ), where each dataset is indexed by an R-tree-based structure. In addition, we have proposed a recursive non-incremental branch-and-bound algorithm following a Depth-First search for processing synchronously all inputs without producing any intermediate result (MPSR). To the best of our knowledge, this is the first algorithm that solves this new and complex distance-based query. The most important conclusions drawn from our experimental study using real spatial datasets are the following: (1) The smaller the R*-tree node size is, the faster the MPSR algorithm becomes. (2) The response time of the query is more affected than the number of disk accesses with the increase of n for a given K, mainly due to the necessary number of distance computations; a similar behavior is obtained from the increase of K for a given n.

Future work may include: (1) The extension of our recursive non-incremental branch-and-bound algorithm to K-Self-MWD Join Query, Semi-MWD Join Query, as well as, the development of algorithms for finding the K $\mathrm{D}_{distance}$-largest n-tuples. (2) The use of approximation techniques in our MPSR algorithm that reduce the search space and produce approximate results faster than the precise ones. Further, the study of the trade-off between cost and accuracy of the results obtained by such techniques.

References

1. N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger; "The R*-tree: and Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD Conf.*, pp. 322–331, 1990.
2. T. Brinkhoff, H.P. Kriegel and B. Seeger; "Efficient Processing of Spatial Joins Using R-trees", *Proc. ACM SIGMOD Conf.*, pp. 237–246, 1993.
3. P. Brown; *Object-Relational Database Development: A Plumber's Guide*, Prentice Hall, 2001.
4. A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos; "Closest Pair Queries in Spatial Databases", *Proc. ACM SIGMOD Conf.*, pp. 189–200, 2000.
5. A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos; "Algorithms for Processing K Closest Pair Queries in Spatial Databases", *Technical Report*, Department of Informatics, Aristotle University of Thessaloniki (Greece), 2001.
6. Digital Chart of the World: Real spatial datasets of the world at 1:1,000,000 scale. 1997 (http://www.maproom.psu.edu/dcw).
7. A. Guttman; "R-trees: A Dynamic Index Structure for Spatial Searching", *Proc. ACM SIGMOD Conf.*, pp. 47–57, 1984.
8. G.R. Hjaltason and H. Samet; "Incremental Distance Join Algorithms for Spatial Databases", *Proc. ACM SIGMOD Conf.*, pp. 237–248, 1998.
9. N. Mamoulis and D. Papadias; "Integration of Spatial Join Algorithms for Processing Multiple Inputs", *Proc. ACM SIGMOD Conf.*, pp. 1–12, 1999.
10. N. Mamoulis and D. Papadias; "Multiway Spatial Joins", *ACM Transactions on Database Systems (TODS)*, Vol. 26, No. 4, pp. 424–475, 2001.

11. Y. Manolopoulos, Y. Theodoridis and V. Tsotras; *Advanced Database Indexing*, Kluwer Academic Publishers, 1999.
12. Oracle Technology Network; *Oracle Spatial*, an Oracle Technical White Paper, 2001 (http://otn.oracle.com/products/oracle9i/pdf/OracleSpatial.pdf).
13. H.H. Park, G.H. Cha and C.W. Chung; "Multi-way Spatial Joins Using R-Trees: Methodology and Performance Evaluation", *Proc. 6th Int. Symp. of Spatial Databases (SSD)*, pp. 229–250, 1999.
14. D. Papadias, N. Mamoulis and Y. Theodoridis; "Processing and Optimization of Multiway Spatial Joins Using R-Trees", *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pp. 44–55, 1999.
15. F.P. Preparata and M.I. Shamos; *Computational Geometry: An Introduction*, Springer-Verlag, 1985.
16. H. Shin, B. Moon and S. Lee; "Adaptive Multi-Stage Distance Join Processing", *Proc. ACM SIGMOD Conf.*, pp. 343–354, 2000.

Rule-Based Generation of XML DTDs from UML Class Diagrams

Thomas Kudrass and Tobias Krumbein

Leipzig University of Applied Sciences,
Department of Computer Science and Mathematics, D-04251 Leipzig
{kudrass|tkrumbe}@imn.htwk-leipzig.de

Abstract. We present an approach of how to extract automatically an XML document structure from a conceptual data model that describes the content of a document. We use UML class diagrams as the conceptual model that can be represented in XML syntax (XMI). The algorithm we present in the paper is implemented as a set of rules that transform the UML class diagram into an adequate document type definition (DTD). The generation of the DTD from the semantic model corresponds with the logical XML database design with the DTD as the database schema description. Therefore, we consider many semantic issues, such as the dealing with relationships, how to express them in a DTD in order to minimize the loss of semantics. Since our algorithm is based on XSLT stylesheets, its transformation rules can be modified in a very flexible manner in order to consider different mapping strategies and requirements.

Keywords: UML, DTD, XML, Schema Generation

1 Introduction

Conceptual modeling of information is a widely accepted method of database design. It improves the quality of the databases, supports an early recognition of design errors and reduces the cost of the development process. A conceptual schema facilitates the communication with the domain expert since it abstracts from the implementation. Due to the analogy of the relational database design we must embrace a 3-level information architecture for XML databases, also known as document viewpoints [1]. This architecture allows the data modeler to start by focusing on conceptual domain modeling issues rather than implementation issues. At the conceptual level, the focus is on data structures, semantic relationships between data and integrity constraints (information viewpoint). The information of an XML document can be arranged in a logical structure (logical level) and is stored dependent on the type of the document (physical level).

Currently, DTDs are the most common way to specify an XML document schema, which corresponds with the logical structure of the document. The textual description of a DTD facilitates the communication in the WWW and the processing with XML parsers. There is a number of tree-based graphical tools for developing the document structure, such as XML Spy or XML Authority. But there are almost no established methods that explicitly model the information of an XML document at the conceptual level. The more complex the data is, the harder is it for the designer to produce the correct document schema. UML makes it easier to visualize the conceptual model and to express the integrity constraints.

There are only a few publications on the automatic generation of XML document schemas from conceptual models. Conrad et al. [2] propose a set of transformation rules for UML class diagrams into XML DTDs. In this paper, UML associations are only translated into XLinks and there is no complete algorithm available. Another approach is to extract semantic information from the relational database schema as it is proposed in [3]. The authors ignore many semantic issues like cardinality or key constraints. In [4] the authors propose an algorithm for the automatic generation of XML DTDs from an (Extended) Entity Relationship Diagram. They intend an implementation by reconstructing the ER schema from a relational database schema. Another interesting approach is presented in [5] that describes a mapping of UML class diagrams into XML Schema definitions using the 3-level design approach. They represent the logical design level by UML class diagrams, which are enhanced by stereotypes to express the XML Schema facilities. EER schemas and UML class diagrams have much in common, which makes it possible to adapt mapping procedures from both source models for the generation of DTDs. On one hand, there is a variety of mapping strategies for the logical XML database design. On the other hand, there are almost no reports on working implementations. This paper contributes a mapping algorithm for the automatic generation of DTDs using stylesheets to represent the transformation rules. Our approach is open since the algorithm is adaptable by changing rules. In the same way, the rules can be applied to the generation of another target schema, such as XML Schema or even a relational schema.

This paper is organized as follows: Section 2 gives an overview of UML class diagrams that are used for modeling data structures. For every diagram element, different mapping strategies are discussed, which can be expressed in transformation rules to generate an adequate DTD representation. Section 3 is an overview about the complete algorithm for the generation of DTDs from UML class diagrams that are implemented as rules. This algorithm is illustrated on a sample model. Then the implementation with XSLT - based on XMI - and the rules of the XSLT stylesheet are described. Options and limitations of the mapping approach in section 4 are discussed. As a conclusion, the assessment of the experiences are given in section 5.

2 Mapping UML Class Diagrams into XML Structures

2.1 Elements of UML Class Diagrams

The primary element of class diagrams is the class. A class definition is divided into three parts: class name (plus stereotypes or properties), attributes and operations of the class. A class can be an abstract one. Attributes can be differentiated into class attributes (underlined) and instance attributes. An attribute definition consists of: visibility (public, protected, private), attribute name, multiplicity, type, default value and possibly other properties. Derived attributes can be defined, i.e. their values can be computed from other attribute values. They are depicted by a '/' prefix before the name. UML types can be primitive or enumeration types or complex types. Classes can be arranged in a generalization hierarchy which allows multiple inheritance.

Associations describe relationships between classes in UML, which are represented by lines, for example an association between classes A and B. The multiplicity r..s at the B end specifies that an instance of A can have a relationship with at least r instances and at most s instances of B. Associations can be comprised more than two classes. Those n-ary associations are represented by a rhomb in the diagram. Associations can be marked as navigable, which means that the association can be traversed only along in one direction. Yet the default is a bidirectional association. In order to specify attributes of an association, an association class has to be defined additionally.

Besides general associations UML provides special types of associations. Among them is the aggregation representing a part-of semantics (drawn by a small empty diamond in the diagram). The composition as another type is more restrictive, i.e., a class can have at most one composition relationship with a parent class (exclusive) and its life span is coupled with the existence of the super class. It is represented by a black diamond at the end of the composite class. Qualified association is a special type of association. Qualifiers are attributes of the association, whose values partition the set of instances associated with an instance across an association.
The elements of an UML model can be modularized and structured by the usage of packages.

2.2 Mapping of Classes and Attributes

UML classes and XML elements have much in common: Both have a name and a number of attributes. Hence a class is represented by an element definition; operations do not have an XML equivalent. The generated XML element has the same name as the UML class. The elements need to be extended by an ID attribute in order to refer them from other parts of the document. Note that the object identity applies only within the scope of one document. Abstract classes should be mapped to parameter entities to support the reuse of their definitions by their subclasses. It is also possible to define an element for an abstract class without declaring it within the package element.

Classes with the stereotype enumeration are separately handled as enumeration datatypes. For all attributes that have an enumeration class as a datatype, an enumeration list is defined with the attribute names of the enumeration class as values. Other stereotypes are represented as prefixes of the element name or attribute name.
UML attributes can be transformed into XML attributes or subelements. A representation as XML attribute is restricted to attributes of primitive datatypes and therefore not applicable to complex or set-valued attributes. A workaround solution is the usage of the NMTOKENS type for XML attributes, although this excludes attribute values containing blanks. A default value, a fixed value and a value list in a DTD can be assigned to attributes which is not possible for elements.

Table 1. Attributes vs. elements at DTD generation

UML element	XML attribute	XML element
primitive datatypes	supported	supported
complex datatypes	not supported	supported
Multiplicity	[0..1] and [1..1]	all
property string	not supported	supported
default value	default property	not supported
fixed value	#FIXED ’value’	not supported
value list	enumeration supported	not supported
scope of definition	local	global

The last entry of Table 1 highlights a serious problem for an automatic transformation of UML attributes into elements. XML attributes are always defined within a certain element, whereas elements are globally defined within the whole document. Therefore name conflicts may occur when UML attributes are transformed into XML elements.

There are some UML constructs which cannot be translated into an adequate document type definition: The visibility properties of UML attributes cannot be transformed due to the lack of encapsulation in XML. The property {*frozen*} determines that an attribute value can be assigned once and remains static, which cannot be mapped properly to an equivalent XML construct. The only workaround solution is to define an initial value as default with the property *fixed* in a DTD. Class attributes are also not supported in XML; they can be marked by naming conventions in the automatic transformation. An adequate transformation of derived attributes into XML would require access to other document parts which implies the transformation of the derivation expression into an XPath expression. Derived attributes are ignored because they do not carryinformation.

2.3 Mapping of Associations

2.3.1 Approaches for Binary Associations

The most crucial issue of the transformation algorithm is the treatment of UML associations. There are different procedures on how to represent associations in a DTD but all of them result in some loss of information regarding the source model. There are four approaches, which are subsequently discussed.

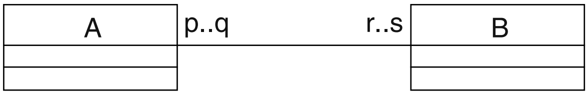


Fig. 1. Mapping of non-hierarchical relationships

- nested elements (hierarchical relationship)
- ID/IDREF references of elements
- references via association element
- references with XLink and Xpointer

Hierarchical Relationship

The hierarchical relationship is the "natural" relationship in XML because it corresponds with the tree structure of XML documents. Elements are nested within their parent elements which implies some restrictions. The existence of the subelement depends on the parent element. If B is represented as subelement of A, the upper bound of its multiplicity q is restricted to 1. Usually p must also be 1. Otherwise, alternative mappings have to be defined, e.g. the definition of B as subelement of the root element. The main obstacle for the nesting of elements is the creation of redundancies in case of many-to-many relationships. It depends on the application profile as to how far redundancy in the document can be tolerated. For example, read-only applications may accept redundancy within a document because it fastens the access to related information. From the viewpoint of the logical XML database design the hierarchical approach appears inappropriate.

Regarding hierarchical representation, it is also difficult to deal with recursive associations or relationship cycles between two or more classes. The XML documents have a document tree of indefinite depth. This can be avoided by treating each association as optional - regardless of the constraint definition in the class diagram.

ID/IDREF References

The ID/IDREF relationship is expressed by adding an ID attribute to elements to be referenced. The references are implemented by attributes of type IDREF (single reference) or IDREFS (multiple reference). Depending on the multiplicity p..q the reference attribute ref of B is defined as follows:

Table 2. Mapping of multiplicity constraints

p	q	Definition of the reference attribute of B
0	1	`<!ATTLIST B ref IDREF #IMPLIED>`
0	*	`<!ATTLIST B ref IDREFS #IMPLIED>`
1	1	`<!ATTLIST B ref IDREF #REQUIRED>`
1	*	`<!ATTLIST B ref IDREFS #REQUIRED>`

There are serious restrictions, which obstruct a semantically correct mapping. The "type" IDREFS accepts duplicate reference whereas the multiplicity in UML denotes the number of distinct instances of an association. For a better representation of multiple references, it is also possible to define an element with an IDREF attribute and with the multiplicity in the parent element. The main drawback is the lacking type safety in the ID/IDREF representation. IDREF can reference elements of any type. The type information could by expressed by naming conventions for the reference attributes without enforcing the integrity. Bidirectional associations are represented by two ID/IDREF references in the DTD. However, this approach cannot guarantee a mutual reference between two element instances that take part in a bidirectional association.

References via Association Elements

For each association an association element is introduced that references both participating elements using IDREF attributes (analogous to relations for many-to-many relationships in RDBMS). The association elements are included as subelements of the document root. There are no references in the class elements. The association element gets the name of the association, the references are labeled according to the association roles. The approach produces XML documents with minimal redundancy, because every instance needs to be stored only once within the document.

The multiplicity values cannot be expressed adequately by association elements. We can merely define how many elements are related by an association instance. This does not consider participation constraints for the element instances. Association elements are particularly useful for n-ary associations and attributed associations only, because of their limitations.

References with XLinks

XLinks have been invented for hyperlink documents that are referencing each other, which makes it possible to reference different document fragments. The extended features provided by XLinks have been considered. The association element is represented as *extended* link. A *locator* element is needed for each associated element to identify it. The association itself is established by *arc* elements that specify the direction. The use of XLinks has been explored by [2]. However, this approach has no type safety.

2.3.2 Association Classes

An association class is an association with class features. So the transformation has to consider the mapping of both a class and an association. Therefore, the four mapping approaches for associations, as sketched above, apply for association classes as well.

The association class is mapped to an association element that is nested towards the parent element in the hierarchical approach (for functional relationships only). The association attributes and the child element in the hierarchical approach are added to the association element.

Using ID/IDREF references requires the introduction of two references to consider bidirectional relationships. Thus, the attributes of the association class would be stored twice. It could not be guaranteed that those attributes are the same in two mutually referencing elements. Thus the mapping has to be enhanced by an association element.

The association elements contain the attributes of the corresponding association class. Associations of each multiplicity are dealt with the same way.

References with extended XLinks is comparable with association elements with the same conclusion as mentioned above.

It is also possible to resolve the association class and represent it as two separate associations. Note that the semantics of bidirectional associations cannot be preserved adequately with that mapping.

2.3.3 N-ary Associations

N-ary associations can also be treated by using one of the four mapping approaches for associations. Simple hierarchical relationships or ID/IDREF references are not appropriate; they support binary associations at best. Better mappings are association elements and extended XLinks, because they can contain the attributes of n-ary associations and represent an association with references to all association ends. Alternatively, the n-ary association can be resolved into n binary associations between every class and the association element.

2.3.4 Other Properties of Associations / Limitations

Each end of an association can be assigned the *{ordered}* property to determine the order of the associated instances. It is not possible to define the order of element instances in a DTD.

The direction of an association cannot be preserved by mapping approaches that represent just bidirectional associations. This applies to: hierarchical relationships, association elements, extended XLinks.

UML provides association properties regarding changeability: *{frozen}* and *{addonly}*. Addonly allows an instance to join more associations instances without deleting or changing existing ones. Both properties cannot be expressed in XML.

There are no means to represent access properties of associations in XML.

In UML, a qualifier can be defined at an association end to restrict the set of instances that can take part in the association. The described mapping procedures do not support qualifiers as they cannot guarantee type safety.

XOR constraints specify the participation of an instance in one of many possible associations in UML. In a DTD, one can define alternative subelements, listed by |. When exporting the UML class diagram into XMI with the *Unisys Rose XML Tool* the XOR constraints are represented only as comments in XMI, which are split up among different elements. So the information about the UML elements related by the constraint cannot be preserved during the transformation.

2.4 Mapping of Generalization

There is no generalization construct in the DTD standard. The most relevant aspect of generalization is the inheritance of attributes of the superclass. There are two reasonable approaches to represent inheritance in the DTD: parameter entities and embedded elements. Parameter entities are defined for attributes and subelements of superclasses. They are inherited by the subclass using parameter entities in the definition of the corresponding element in XML. Alternatively, the superclass element can be embedded completely into the subclass element. To express the substitution relationship between a superclass and its subclasses, the use of a superclass element is substituted by a choice list that contains the superclass element and all its subclass elements. Another solution is the embedding of the subclasses into the superclass. The best way to represent a superclass with different subclasses is to use a choice list in the element definition of the superclass. So the subclass element is nested within the superclass element giving up its identity.

2.5 Further Mapping Issues

The aggregation relationship of UML embodies a simple part-of semantics whereas the existence of the part does not depend on the parent. Therefore aggregations are treated like associations.

Compositions can be mapped through hierarchical relationships according to the previous proposal for associations, because nested elements are dependent on the existence of their parent elements and, hence, represent the semantics of compositions.

Packages are represented as elements without attributes. The name of the element is the package name. All elements of the classes and packages are subelements of their package element.

3 Generation of DTDs from Class Diagrams

3.1 Algorithm

Among different alternatives, discussed in the section above, an overview is given about the transformation methods which have been implemented as rules instead of a conventional algorithm (for further details see [6]).

Table 3. Mapping of UML elements to DTDs

UML Element	XML DTD
class	element, with ID attribute
abstract class	element but not subelement of the parent element
attribute	attribute of the corresponding class element
stereotype	prefix of the element name or attribute name
package	element without attributes
association	reference element, with IDREF attribute referencing the associated class
association class	association class element with IDREF references to both associated classes (resolve the association class)
qualified association	currently not mapped
aggregation	like association
composition	reference element, with subordinated class element (hierarchical relationship)
generalization	superclass is nested within subclass element
association constraint	currently not mapped
n-ary association	association element with IDREF references to all associated classes (resolve the n-ary association)

3.2 Sample Model

The following UML example (figure 2) illustrates the transformation algorithm. There is an abstract superclass `Person` as generalization of `Employee` and `Manager`, all of them belong to the package `People`. The model contains several bidirectional associations: a one-to-one relationship between `Manager` and `Department`, a one-to-many relationship between `Department` and `Employees`, a many-to-many relationship between `Employees` and `Projects` as well as a unidirectional relationship between `Department` and `Project`. The association between `Company` and `Employees` is an attributed one-to-many relationship that is represented by the association class `Contract`. Furthermore, a `Company` is defined as a composition of 1..n `Departments`.

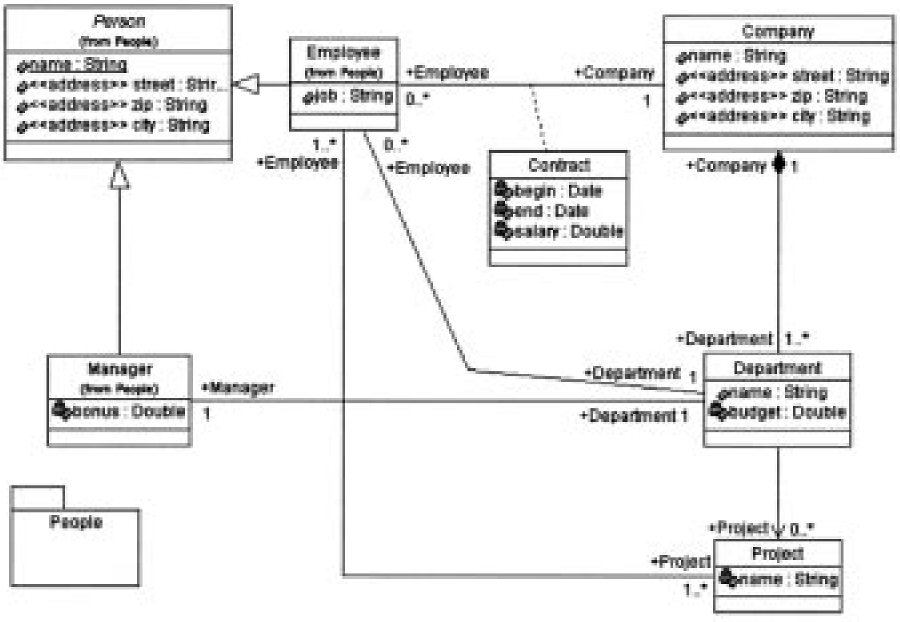


Fig. 2. UML class diagram of sample model

```
<!ELEMENT sample (People, Company*, Project*, Contract*)>
<!ELEMENT People (Employee*, Manager*)>
<!ELEMENT Person EMPTY>
<!ATTLIST Person
  id ID #REQUIRED
  name CDATA #REQUIRED
  address.street CDATA #REQUIRED
  address.zip CDATA #REQUIRED
  address.city CDATA #REQUIRED>
<!ELEMENT Employee (Person, Ref_Employee.Department,
  Ref_Employee.Project+, Ref_Employee.Company)>
```

```
<!ATTLIST Employee
  id ID #REQUIRED
  job CDATA #REQUIRED>
    <!ELEMENT Ref_Employee.Department EMPTY>
    <!ATTLIST Ref_Employee.Department
      Department IDREF #REQUIRED>
    <!ELEMENT Ref_Employee.Project EMPTY>
    <!ATTLIST Ref_Employee.Project
      Project IDREF #REQUIRED>
    <!ELEMENT Ref_Employee.Company EMPTY>
    <!ATTLIST Ref_Employee.Company
      Contract IDREF #REQUIRED>
<!ELEMENT Manager (Person, Ref_Manager.Department)>
<!ATTLIST Manager ... >
    <!ELEMENT Ref_Manager.Department EMPTY>
    <!ATTLIST Ref_Manager.Department
      Department IDREF #REQUIRED>
<!ELEMENT Company (Ref_Company.Employee*,
  Ref_Company.Department+)>
<!ATTLIST Company ... >
    <!ELEMENT Ref_Company.Employee EMPTY>
    <!ATTLIST Ref_Company.Employee
      Contract IDREF #REQUIRED>
    <!ELEMENT Ref_Company.Department (Department)>
<!ELEMENT Department (Ref_Department.Employee*,
  Ref_Department.Manager, Ref_Department.Project*)>
<!ATTLIST Department ... >
    <!ELEMENT Ref_Department.Employee EMPTY>
    <!ATTLIST Ref_Department.Employee ... >
    <!ELEMENT Ref_Department.Manager EMPTY>
    <!ATTLIST Ref_Department.Manager ... >
    <!ELEMENT Ref_Department.Project EMPTY>
    <!ATTLIST Ref_Department.Project ... >
<!ELEMENT Project (Ref_Project.Employee+)>
<!ATTLIST Project ... >
    <!ELEMENT Ref_Project.Employee EMPTY>
    <!ATTLIST Ref_Project.Employee ... >
<!ELEMENT Contract (Ref_Contract.Company,
  Ref_Contract.Employee)>
<!ATTLIST Contract ... >
    <!ELEMENT Ref_Contract.Company EMPTY>
    <!ATTLIST Ref_Contract.Company
      Company IDREF #REQUIRED>
    <!ELEMENT Ref_Contract.Employee EMPTY>
    <!ATTLIST Ref_Contract.Employee
      Employee IDREF #REQUIRED>
```

3.3 Implementation

The XMI format (*XML Metadata Interchange*) makes it possible to represent an UML model in an XML format. This implementation is based on the XMI version 1.1 [7]. The XMI standard describes the generation of DTDs from a meta model as well as the generation of an XMI document from any model, provided they are MOF compliant (*Meta Object Facility*).

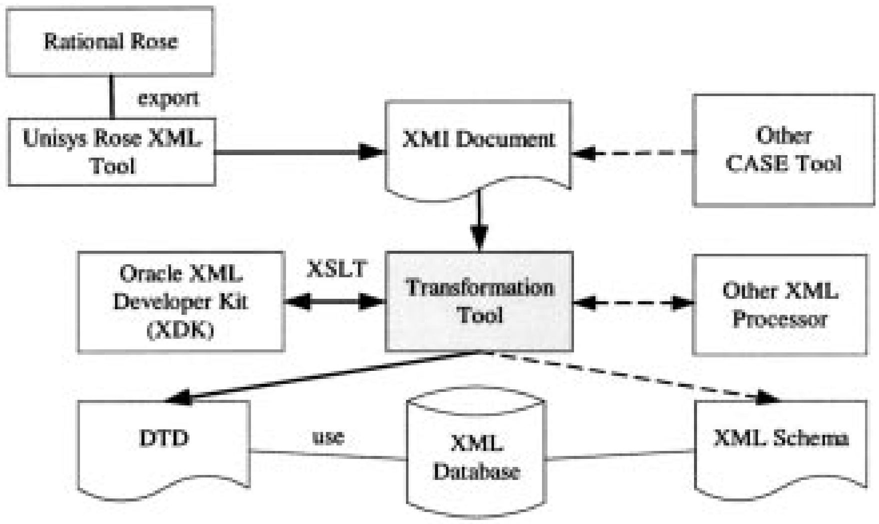


Fig. 3. Overall structure of the DTD generation

UML class models are edited with the CASE tool Rational Rose. The model information can be stored in XMI documents using the *Unisys Rose XML Tool* as an extension. Since XMI is a standard, the original tool is not relevant for the next transformation steps.

The actual transformation is implemented with XSLT (*eXtensible Stylesheet Language Transformation*) that can process the XMI document as any other XML document. XSLT is a language to transform XML documents into other XML documents or even other formats. The stylesheet document consists of rules that specify how the document tree of the source document has to be transformed into the target tree. The rules called template rules have two parts: a search pattern (source tree) and a template applied for matching patterns.

In this implementation, there are two categories of template rules: Some template rules have a pattern that must match with certain XMI elements that are relevant for the conceptual data model. Among them is the `UML:Class` template that transforms a UML class description into the corresponding element definition in the DTD. Some other templates are just auxiliary templates without matching XMI elements. Instead, they are invoked by other templates that make use of their functionality. The transformation program starts with the root template rule. Subsequently the template rules are shown as they are currently implemented.

/ (Root Template)
The root template is called first. It checks the XMI version, determines the first model element and calls the next matching template.

UML:Model | UML:Package
Because `UML:Model` and `UML:Package` elements have the same structure and are transformed the same way, they are combined in the same template. The `UML:Model` element is the root element of the UML model and comprises all other elements like UML packages. For each package an element type is created. The name of the element corresponds with the complete package name. Possible stereotypes appear as a prefix before the package name. The definition of the package element is completed by the list of names of all packages, classes (that are not abstract and not parts of another class in a composition), association classes and n-ary associations that are the topmost level within the package. Afterwards, for each subelement of the package element the appropriate template is activated.

UML:Class
For each `UML:Class` element an element type is defined in the DTD. The name of the element corresponds with the class name. The name of a possible stereotype appears as prefix before the class name. Next, the content of the class element - i.e., all superclasses, complex attributes and associations – are extracted. They are determined by an XPath query on the XMI document. For example, the superclasses are represented in the `UML:Generalization.parent` element. The superclass is determined by the reference represented by the `xmi.idref` attribute. The superclass appears as a subelement of the current element. An element with the name of the attribute and the class name as a prefix before the attribute name is defined for all complex attributes. In XMI, the associations of a class cannot be found within the class element. Instead, they have to be queried throughout the whole XMI document where they are represented as association elements. Once an association of a class has been found it is processed by calling the `Multiplicity` template. This template needs the name of the association element and the cardinality as parameters. The chosen naming convention for association elements is: `Ref_`*`Classname.Rolename`*. In the third step, all simple datatype attributes of a class are defined. Each class receives an ID attribute to make it a potential target of element references. The attributes of a class can be extracted from the `UML:Attribute` elements. Finally, the templates `createComplexAttrib-uteElements` and `createAssociationElements` are called to define the necessary complex elements and reference elements with the IDREF attribute. Those reference elements have been included into the class definition and are defined by this template.

UML:AssocationClass
This algorithm transforms an association class into a class and two associations. So it works the same way as the `UML:Class` template. In addition, two or more associa-

tions have to be defined for each associated class involved in it. The attributes of association classes are treated like attributes of usual classes.

UML:Association

This template is exclusively called by n-ary associations because only these associations are embedded in a package element. It defines an element for the n-ary association with the name of this association and associations for each associated class involved in it.

Multiplicity

Unlike the other templates above, this template does not process the content of the XMI document but the content of the three received parameters that control the transformation. They are: the name of the reference element of the association, the lower and the upper bound of the cardinality. The transformation of the cardinality is based on the rules represented in table 4. The repeated generation of `ref` attributes is realized by recursive calls of the template.

Table 4. Transformation of cardinality constraints into DTD

Cardinality	Result of Transformation
0..n	`ref*`
1..n	`ref+`
2..n	`(ref, ref+)`
0..1	`ref?`
1..1	`ref`
0..2	`(ref?, ref?)`
1..2	`(ref, ref?)`
m..n	`(ref, ref, ref?, ref?)` Example: m=2, n=4

createComplexAttributeElements

This template is called from the `UML:Class` and the `UML:AssociationClass` templates. It defines an element with the name of the attribute and the class name as a prefix before the attribute name for all complex attributes of a class. The content of this element is the element of the complex datatype.

createAssociationElements

This template is also called from both the `UML:Class` and the `UML:AssociationClass` templates. It determines all associations of a class and defines the reference elements of an association. Those reference elements have been included into the DTD before (cf. `UML:Class` template). For each reference element an IDREF attribute is defined. The name of the attribute is composed of the name of the target class to be referenced. At association classes two more reference elements are generated for the newly created association.

Stereotype
The `Stereotype` template checks for stereotypes for all UML elements. Those are referenced by the `stereotype` element via object IDREFS in XMI.

Name
This template determines the name of the current UML element. The name is stored either in the `name` attribute of the element or in the `UML:ModelElement.name` subelement in the XMI definition.

4 Options and Limitations

A number of options are available when mapping the document definition from the conceptual level to the logical level. Section 2 has already outlined alternatives for most UML elements. It just requires the change of template rules to vary certain transformation steps. For example, by changing the template rules the mapping of UML attributes can be modified. In the same way rules can be substituted to implement alternative mappings for the generalization relationship: Instead of nesting elements, the use of parameter entities can be a viable alternative for an adequate representation in the DTD.

In order to assess the quality of the transformation the loss of information has to be determined. This can be done by a reverse transformation of the generated DTD. The following UML elements could not be represented in the DTD. Therefore they are not considered at the reverse transformation:

- stereotypes of associations, aggregations, compositions, generalizations
- name of associations, aggregations, compositions, generalizations
- dependencies
- type integrity of associations
- qualified associations
- data type of attributes

Dependencies have not been transformed because their definition bases mainly on the class behavior, which cannot be expressed in a DTD. In this implementation, the full syntax of the DTD has not yet been used. Among the elements that also should be included are entities and notations.
When transforming a conceptual data model into a DTD two fundamental drawbacks inherent to the DTD have to be dealt with: DTD supports weak typing only. So only the CDATA type for strings is available. Numeric or other data types cannot be expressed adequately. Accordingly, the right type of document content cannot be guaranteed by an XML DBMS using DTDs.

Another serious drawback is the lacking type safety of references. Neither ID/IDREF nor XLink can define the target type of a reference. The only workaround used was a naming convention for elements and attributes to denote the original relationship. DTD cannot define elements with subelements in an arbitrary order. Furthermore, there are no object-oriented constructs such as generalization or any semantic relationships. The uniqueness constraint for key values cannot be enforced by a definition in a DTD.

Also Rational Rose has some limitations. Therefore it is not possible to define attributes with a multiplicity greater than one and n-ary associations. On the other hand, the multiplicity of the aggregate end of an aggregation or composition can exceed one in Rational Rose.

5 Conclusion

This paper presents a very flexible method for the logical XML database design by transforming the conceptual data model represented in UML. UML was primarily chosen because of its widespread and growing use. Yet it would also be possible to use the extended ER model to describe the XML document at the conceptual level. In this approach, the conceptual model and the XML representation of the document content were strictly separated. Therefore, XML specific constructs in the conceptual model are not involved as they can be found, e.g., in DTD profiles for UML [8] or XML extensions of the ER model [9]. This methodology is well-suited for the storage of data-centric documents exchanged among different applications. Vendors of XML database systems are able to process document schemas when storing the XML documents in the database. So the result of this transformation can easily be combined with an XML DBMS such as Tamino (by Software AG), which accepts DTDs as document schema.

The design of the transformation stylesheets has to consider the interplay of the templates when modifying some of the mapping rules to implement a different strategy. A well-designed set of templates as presented in this paper is the precondition to adapt this transformation tool to other target models as well. Currently we are working on mapping algorithms that produce an XML Schema definition as an output of the transformation process. For that purpose, some of these transformation rules have to be rewritten to consider the extended semantic capabilities beyond those of DTDs.

Acknowledgement. This work has been funded by the Saxonian Department of Science and Art (Sächsisches Ministerium für Wissenschaft und Kunst) through the HWP program.

References

[1] H. Kilov, L. Cuthbert: A model for document management, Computer Communications, Vol. 18, No. 6, Elsevier Science B.V., 1995.

[2] R. Conrad, D. Scheffner, J.C. Freytag: XML Conceptual Modeling Using UML, Proc. Conceptual Modeling Conference ER2000, Salt Lake City, USA, Springer Verlag, 2000, pp. 558–571.

[3] G. Kappel, E. Kapsammer, S. Rausch-Schott, W. Retschitzegger: X-Ray – Towards Integrating XML and Relational Database Systems, Proc. 19th Conference on Conceptual Modeling (ER2000), Salt Lake City, 2000.

[4] C. Kleiner, U. Liepeck: Automatic generation of XML-DTDs from conceptual database schemas (in German), Datenbank-Spektrum 2, dpunkt-Verlag, 2002, pp. 14-22.

[5] N. Routledge, L. Bird, A. Goodschild: UML and XML Schema, Proc. 13th Australasian Database Conference (ADC2002), Melbourne, 2002.

[6] T. Krumbein: Logical Design of XML Databases by Transformation of a Conceptual Schema, Masters Thesis (in German), HTWK Leipzig, 2003, available at tkrumbe@imn.htwk-leipzig.de.

[7] OMG: XML Metadata Interchange, http://www.omg.org/cgi-bin/doc?formal/00-11-02.pdf, 2001.

[8] D. Carlson: Modeling XML Applications with UML: Practical E-Business Applications, Boston, Addison Wesley, 2001.

[9] G. Psaila: ERX – A Conceptual Model for XML Documents, Proc. of the ACM Symposium of Applied Computing, Como, 2000.

More Functional Dependencies for XML

Sven Hartmann and Sebastian Link

Information Science Research Centre
Massey University, Palmerston North, New Zealand

Abstract. In this paper, we present a new approach towards functional dependencies in XML documents based on homomorphisms between XML data trees and XML schema graphs. While this approach allows us to capture functional dependencies similar to those recently studied by Arenas/Libkin and by Lee/Ling/Low, it also gives rise to a further class of functional dependencies in XML documents. We address some essential differences between the two classes of functional dependencies under discussion resulting in different expressiveness and different inference rules. Examples demonstrate that both classes of functional dependencies appear quite naturally in practice and, thus, should be taken into consideration when designing XML documents.

1 Introduction

The Extensible Markup Language (XML) [7] is nowadays commonly used as a prime format for publishing and exchanging data on the web. In most cases, so far, a data provider stores data in a relational database and transforms the data to XML whenever necessary. As long as XML is used as a mere markup language according to its original intention there is in general no need for persistent storage of XML documents. In addition, however, there is an increasing number of applications using native XML documents. Often, the targets of modern information systems are much too complex to be stored in relational tables with semi-structured data e.g. from molecular biology being the most prevalent example. There is an ongoing trend to use XML for representing these new forms of data as XML provides a simple, universal and widely accepted standard for describing complex data structures.

If XML is expected to serve as a first class data model, effective means for the management of persistent XML documents as databases are needed. As is true for managing traditional data, the management of XML data requires capabilities to cope with integrity, consistency, data independence, recovery, redundancy, views, access rights, integration, and the enforcement of standards and normal forms. Unfortunately, XML itself provides only syntax but does not carry the semantics of the data. However, in relational databases, integrity constraints have been proven useful in making a good design, in preventing update anomalies, and in allowing the application of efficient methods for data storage and access, and for query optimization. Understanding this situation, several classes of integrity constraints have been extended to or newly defined for XML including key dependencies, inclusion dependencies, inverse constraints, and path constraints. In addition, properties such as axiomatization and satisfiability have been studied for these constraints. For details, see [1,3,8–11,17].

Functional dependencies other than key dependencies have been little investigated for XML so far. From relational databases, it is well-known that inserting or modifying data in the presence of a functional dependency is a major source of update anomalies due to redundant data [5,6]. Consequently, functional dependencies have been most widely studied in the literature on the relational database model and are most widely used in practice, cf. [2]. In particular, popular normal forms such as 3NF or BCNF avoiding important kinds of data redundancy are defined on the basis of functional dependencies. In view of this experience it is rather astonishing that functional dependencies so far did not receive adequate attention in the literature on XML. In fact, there are only two papers [4,13] addressing these constraints. Both define functional dependencies on the basis of paths in XML documents. This approach goes back to earlier research on functional dependencies in semantic and object-oriented data models [15,18]. As we will see later on, there are functional dependencies in XML documents which are not captured by this approach.

Our objective is to define functional dependencies on the basis of subgraphs of XML data trees and XML schema trees. This approach is a natural generalization of functional dependencies in relational databases. In addition, it allows us to specify functional dependencies similar to those studied by Arenas/Libkin [4] and Lee/Ling/Low [13] as well as further functional dependencies for XML which appear quite naturally in practice.

The present paper is organized as follows. In Section 2, we give examples of functional dependencies to be discussed within this paper. In Section 3, we assemble some basic terminology to be used later on. In particular, we provide the notions of XML graphs and v-subgraphs. On the basis of these notions, we propose two definitions of functional dependencies for XML in Section 4. In Section 5, we study interactions of functional dependencies for XML and discuss inference rules.

2 An Example

Consider the XML data tree in Fig. 1 containing data on courses organized by the dancing club of the local high school. (A precise definition of XML data trees will be given in the subsequent section. The XML document corresponding to this XML data tree is shown in Fig. 14 in the appendix.) It happens that neither gentlemen nor ladies change their dance partners, that is, for every pair in the XML data tree He determines She, and vice versa. Both observations are likely to be called functional dependencies.

Now consider the XML data tree in Fig. 2. It is obvious that the observed functional dependencies do no longer hold. Nevertheless the data stored in this tree is not independent from each other: whenever two courses coincide in all their pairs then they coincide in their rating, too. That is, in every course the set of Pairs determines the Rating. The reason for this might be straightforward.

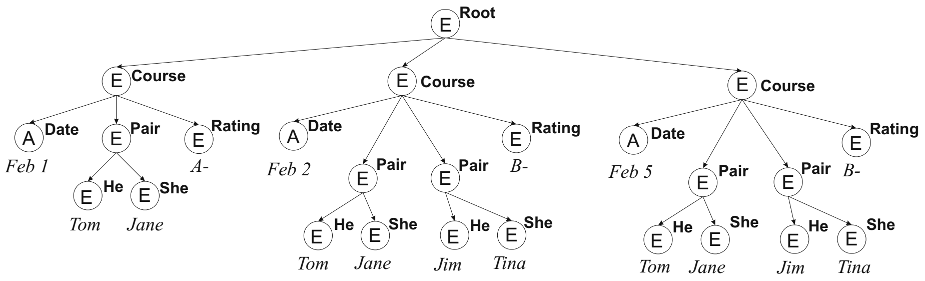


Fig. 1. An XML data tree carrying some functional dependency.

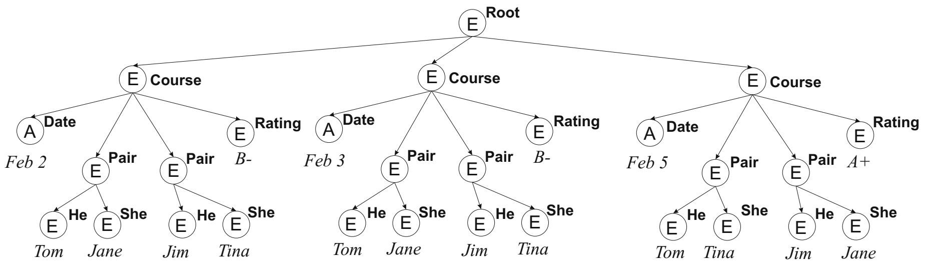


Fig. 2. Another XML data tree still carrying some functional dependency.

Suppose, during every course each pair is asked whether they enjoyed dancing with each other (and suppose that the answer will not change over time). Afterwards, the average rating is calculated for the course and stored within the XML document. This, in fact, leads to the functional dependency observed in Fig. 2.

Surprisingly, both Arenas/Libkin [4] as well as Lee/Ling/Low [13] introduced the first kind of functional dependencies for XML, while the second kind has been neglected so far in the literature on XML. The reason for this is the path-based approach towards functional dependencies used in both papers. This motivated us to investigate a subgraph-based approach which allows us to cope with both kinds of functional dependencies.

3 Graphs and Trees for XML

Within this paper we use a fairly simple XML graph model, which is still powerful enough to point out essential differences between both kinds of functional dependencies under discussion. We assume that the reader is familiar with basic notions from graph theory such as graphs, trees and paths. For a comprehensive introduction to graph theory, we refer to [12]. Note that all graphs within this paper are considered to be finite, directed and without parallel arcs.

3.1 Rooted Graphs and Trees

A *rooted graph* is a directed acyclic graph $G = (V_G, A_G)$ with one distinguished vertex r_G, called the *root* of G, such that every vertex of G can be reached from r_G by passing a directed path of forward arcs. In a rooted graph every vertex but r_G has at least one predecessor, while r_G has no predecessor. For every vertex v, let $Succ_G(v)$ denote its (possibly empty) set of successors in G. Vertices without successors are said to be *leaves*. Let $L_G \subseteq V_G$ denote the set of leaves of G. A *rooted tree* is a rooted graph $T = (V_T, A_T)$ where every vertex $v \in V_T$ but the root r_T has exactly one predecessor.

3.2 XML Graphs and Trees

Rooted graphs are frequently used to illustrate the structure of XML documents, see e.g. [1,4,9,11,14]. In general, we suppose that there are fixed sets $ENames$ of element names and $ANames$ of attribute names, and a fixed symbol S indicating text.

An *XML graph* is a rooted graph G together with two mappings $name : V_G \rightarrow ENames \cup ANames$ and $kind : V_G \rightarrow \{E, A\}$ assigning every vertex its name and kind, respectively. If G is, in particular, a rooted tree we also speak of an *XML tree*. The symbols E and A tell us whether a vertex represents an element or an attribute, respectively. We suppose that vertices of kind A are always leaves, while vertices of kind E can be either leaves or non-leaves. In addition, we suppose that vertices of kind E have a name from $ENames$, while vertices of kind A have a name from $ANames$.

An *XML data tree* is an XML tree T together with an evaluation $val : L_T \rightarrow STRING$ assigning every leaf a (possibly empty) string.

An *XML schema graph* is an XML graph G together with a mapping $freq : A_G \rightarrow \{1, *\}$ assigning every arc its frequency. We suppose that arcs terminating in vertices of kind A always have frequency 1, while arcs terminating in vertices of kind E may have frequency 1 or $*$. In Fig. 5, for example, we marked all arcs of frequency $*$ by a $*$. Further, we suppose that no vertex in an XML schema graph G has two successors sharing both their kind and their name. Hence, the first graph in Fig. 3 may serve as an XML schema graph, while the second one

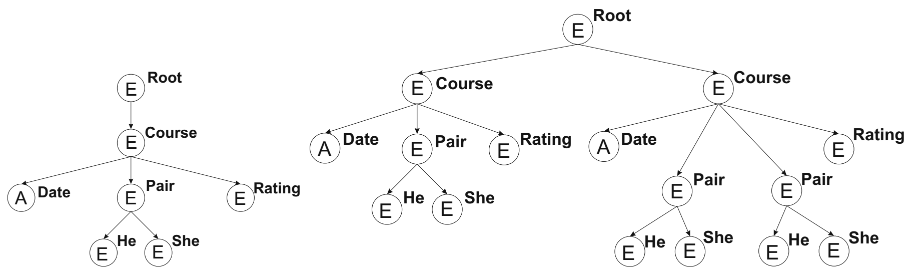


Fig. 3. Two XML trees with vertex labels illustrating names and kinds of vertices.

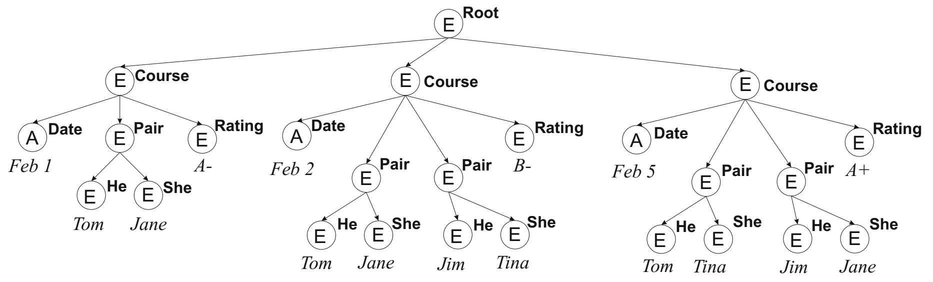


Fig. 4. An XML data tree with additional labels for the leaves illustrating their values.

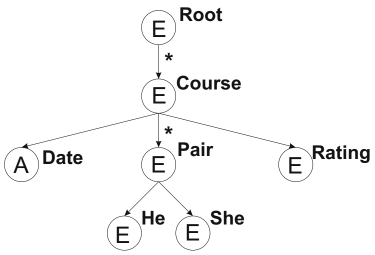


Fig. 5. An XML schema tree with additional labels indicating $*$-frequencies.

does not. If G is, in particular, an XML tree we also speak of an *XML schema tree.*

Note that XML schema graphs should not be mixed up with the popular language XML Schema [16] which can be used to describe XML documents. Instead, we use this term to emphasize the analogy with a database schema in traditional database design.

A further remark concerning the frequencies is called for. Our intention is to use XML schema graphs to describe XML documents. In this sense, the frequency of an arc $a = (v, w)$ between two vertices of kind E tells us whether elements of the XML document modelled by the vertex v may contain many elements modelled by the vertex w or no more than a single one. In the example of Fig. 5, this means that a course may contain several pairs, but only one rating.

It is rather easy to determine the frequencies when deriving an XML schema graph from a DTD. In this case every element, that is, every vertex v of kind E comes along with a regular expression $exp(v)$ over the alphabet $ENAMES$ or is associated with text, that is, $exp(v) = S$. We suppose that the regular expression contains only names of successors of v in G, that is, $exp(v) \in \{name(w) : kind(w) = E \text{ and } w \in Succ_G(v)\}^*$. Consequently, every leaf of kind E is associated with the empty expression λ or with text. As an example, consider the XML schema tree in Fig. 5 and assume we are given the regular expressions $exp(v_{Root}) = Course*$, $exp(v_{Course}) = (Pair*, Rating)$, $exp(v_{Pair}) = (He, She)$ as well as $exp(v_{Rating}) = S$, $exp(v_{He}) = S$ and $exp(v_{She}) = S$ indicating that the latter elements are associated with text. From *exp* we may derive the frequency $freq(a)$ of every arc $a = (v, w)$ between two vertices of kind E: it is $*$ if there is some word in the regular language defined by the expression $exp(v)$ which contains the name of w more than once, and 1, otherwise.

Though the authors of [4,13] assumed the existence of a DTD, their ideas straightforward translate to XML schema graphs. In practice, XML documents do not always possess DTDs, but may be developed either from scratch or using some other description. Therefore, we prefer to use XML schema graphs rather than DTDs for the definition of functional dependencies. The advantage is that XML schema graphs may easily be generated from native XML documents.

3.3 Homomorphisms between XML Graphs

Let G' and G be two XML graphs, and consider a mapping $\phi : V_{G'} \to V_G$. We call the mapping ϕ *root-preserving* if the root of G' is mapped to the root of G, that is, $\phi(r_{G'}) = r_G$. Further, ϕ is *kind-preserving* if the image of a vertex is of the same kind as the vertex itself, that is, $kind(v') = kind(\phi(v'))$ for all $v' \in V_{G'}$. Finally, ϕ is *name-preserving* if the image of a vertex carries the same name as the vertex itself, that is, $name(v') = name(\phi(v'))$ for all $v' \in V_{G'}$. The mapping ϕ is a *homomorphism* between G' and G if the following conditions hold:

(i) every arc of G' is mapped to an arc of G, that is, $(v', w') \in A_{G'}$ implies $(\phi(v'), \phi(w')) \in A_G$,
(ii) ϕ is root-, kind- and name-preserving.

A homomorphism ϕ may be naturally extended to the arc set of G': given an arc $a' = (v', w')$ of G', $\phi(a')$ denotes the arc $(\phi(v'), \phi(w'))$ of G.

As an example, consider the two XML trees in Fig. 3. There is a unique name-preserving mapping ϕ which maps the vertices of the second graph G' to the vertices of the first graph G. (That is, all vertices with name Course in the second graph are mapped to the single vertex with name Course in the first graph, etc.) It is not difficult to check, that this mapping satisfies conditions (i) and (ii) mentioned above, i.e., is a homomorphism.

Let T' be an XML data tree and G an XML schema graph. T' is said to be *compatible* with G if there is a homomorphism $\phi : V_{T'} \to V_G$ between T' and G such that for every arc $a = (v, w)$ of frequency 1 in G its pre-images in T' are mutually vertex-disjoint. Due to our definition of XML schema graphs there is exactly one root-, kind- and name-preserving mapping from $V_{T'}$ to V_G, that is, if the homomorphism ϕ exists then it is unique.

For example, consider the XML data tree T' in Fig. 4 and the XML schema tree T in Fig. 5. Again, the unique name-preserving mapping from $V_{T'}$ to V_T is a homomorphism between T' and T. On comparing both trees and checking the frequencies, it turns out that T' is in fact compatible with T.

A homomorphism $\phi : V_{G'} \to V_G$ between XML graphs G' and G is an *isomorphism* if ϕ is bijective and ϕ^{-1} is a homomorphism, too. Then G' is said to be *isomorphic* to G or a *copy* of G. Moreover, we call two isomorphic XML data trees T' and T *equivalent* if the isomorphism $\phi : V_{T'} \to V_T$ between them

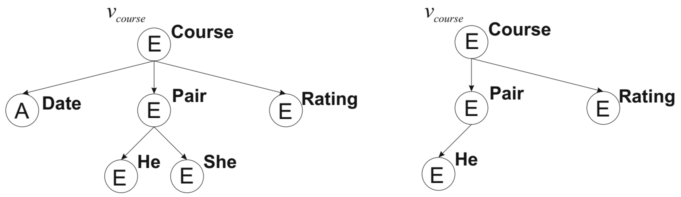


Fig. 6. Two v_{Course}-subgraphs of the XML tree T in Fig. 5, where v_{Course} denotes the vertex with name Course in T. The subgraph on the left hand side is the total v_{Course}-subgraph $T(v_{Course})$ of T.

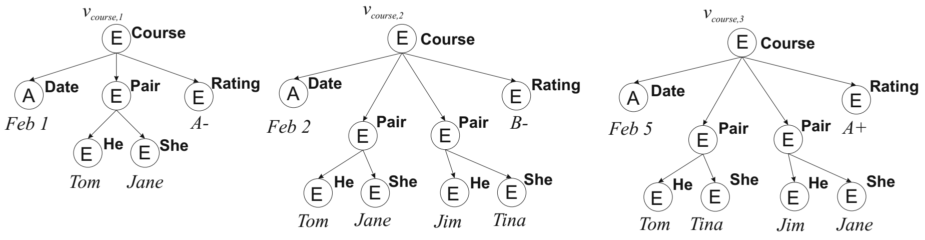


Fig. 7. Three total subgraphs of the XML tree T' in Fig. 4. Their roots $v_{Course,1}$, $v_{Course,2}$ and $v_{Course,3}$ are just the three vertices with name Course in T'. These three total subgraphs are the pre-images of the total v_{Course}-subgraph $T(v_{Course})$ of the XML tree T in Fig. 5 under the unique homomorphism between T' and T.

is also *evaluation-preserving*, that is, $val(\phi(v')) = val(v')$ holds for every vertex $v' \in V_{T'}$.

3.4 v-Subgraphs

Let G be a rooted graph, v be a vertex of G and $L \subseteq L_G$ be a set of leaves of G. Consider the union U of all directed paths of G from v to some leaf in $L = L_U$. We call U a *v-subgraph* of G. Clearly, every non-empty v-subgraph of an XML graph is itself an XML graph. In particular, a non-empty v-subgraph of an XML tree is again an XML tree.

If U contains all leaves of G which may be reached from v by passing a directed path of G, then U is said to be the *total v-subgraph* of G and is denoted by $G(v)$. Note that $G(v)$ is maximal among the v-subgraphs of G and contains every other v-subgraph of G as a v-subgraph itself.

Let G' and G be XML graphs. A homomorphism $\phi : V_{G'} \to V_G$ between them induces a mapping of the total subgraphs of G' to the total subgraphs of G: given a total v'-subgraph $G'(v')$ of G', $\phi(G'(v'))$ denotes the total $\phi(v')$-subgraph $G(\phi(v'))$ of G. Note that the vertices and arcs of $G'(v')$ are mapped to the vertices and arcs of $\phi(G'(v'))$. Obviously, the number of pre-images of a total v-subgraph of G coincides with the number of pre-images of the vertex v.

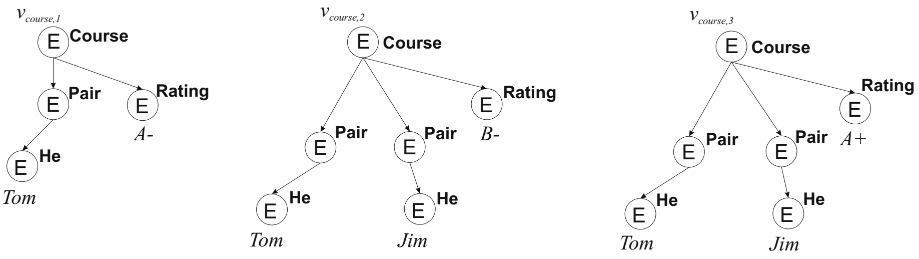


Fig. 8. The projections of the three XML trees in Fig. 7 to the v_{Course}-subgraph U on the right hand side of Fig. 6.

Let G' and G be two XML graphs together with a homomorphism ϕ between them. An $r_{G'}$-subgraph U' of G' is a *subcopy* of G if U' is isomorphic to some r_G-subgraph U of G. A subcopy of G is maximal if it is not contained in any other subcopy of G. Note that copies of G in G' are maximal subcopies, but not vice versa. Maximal subcopies are of special interest as an XML data tree compatible with some XML schema tree T does not necessarily contain copies of T.

3.5 Projections

Let G' and G be XML graphs together with a homomorphism $\phi : V_{G'} \to V_G$ between them. Given an r_G-subgraph U of G, the *projection* of G' to the subgraph U is the $r_{G'}$-subgraph U' of G' whose leaves are just the pre-images of the leaves of U, that is, $L_{U'} = \{u \in L_{G'} : \phi(u) \in L_U\}$. We denote U' by $G'|_U$.

4 Functional Dependencies

We are now ready to discuss possible definitions of functional dependencies for XML. For an XML data tree T', there is usually more than just a single XML schema graph G such that T' is compatible with G. It is well-known that every rooted graph G may be uniquely transformed into a rooted tree T by unfolding it, that is, by splitting vertices with more than one predecessor. For the sake of simplicity, we will restrict ourselves to functional dependencies defined with respect to a fixed XML schema tree T.

A *functional dependency* (or *XFD*, for short) is an expression $v : X \to Y$ where v is a vertex of T, and X and Y are v-subgraphs of T. The natural question to be asked at this moment is: *When does an XML data tree compatible with T satisfy this functional dependency?* In order to give an appropriate answer, we consider a simple example of an XML data tree arising from a straightforward translation of a database relation to XML, see Fig. 9.

The original database relation satisfies the functional dependency Class: Subject, Lecturer $\to$ Room, that is, any two classes on the same subject given by the

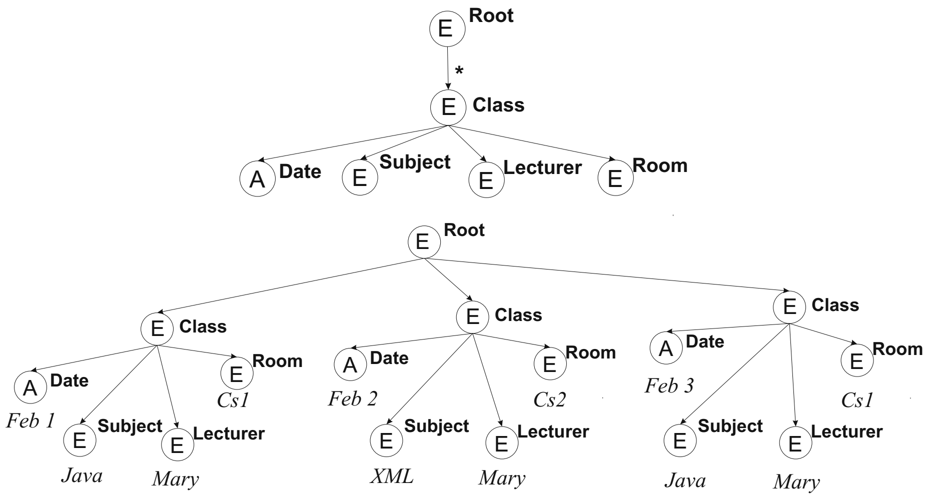


Fig. 9. An XML schema tree and a compatible XML data tree arising from a traditional database relation.

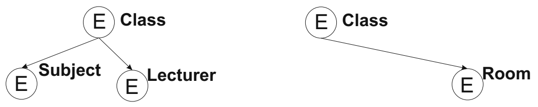


Fig. 10. The v_{Class}-subgraphs X and Y of the XML schema tree in Fig. 9 involved in the XFD $v_{Class} : X \to Y$ under discussion.

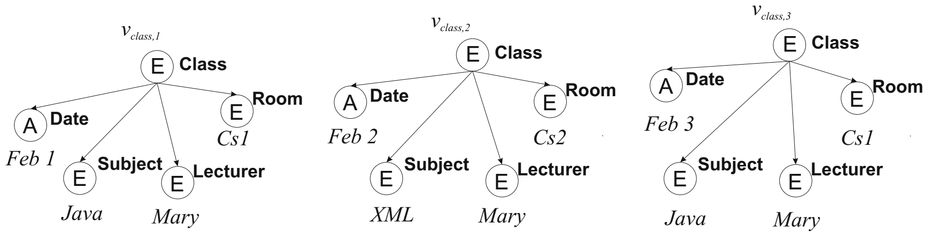


Fig. 11. The three pre-images of the total v_{Class}-subgraph of the XML schema tree in Fig. 9. The first and the third pre-image coincide in their projections to X as well as in their projections to Y.

same lecturer use the same room, too. This translates quite naturally to the XFD $v_{Class} : X \to Y$, where X and Y are the two v_{Class}-subgraphs in Fig. 10. On inspecting the three pre-images of the total v_{Class}-subgraph of the XML schema graph we observe that for the first and the third pre-image their projections to the the subgraph X are equivalent. Caused by the functional dependency under inspection their projections to the subgraph Y are equivalent, too.

The discussion above advises a suitable generalization of functional dependencies to XML. However, it should be emphasized that in the simple example above the pre-images of the total v_{Class}-subgraph are copies of the total v_{Class}-subgraph. In general, this is not the case as shown by the examples in the previous section. This observation gives rise to at least two possible interpretations of a functional dependency for XML:

Functional Dependencies – A First Interpretation: Consider an XML data tree T' which is compatible with T and let ϕ be the unique homomorphism between them. T' *satisfies the XFD* $v : X \to Y$ if for any two **maximal subcopies** W_1 and W_2 of $T(v)$ in T' the projections $W_1|_Y$ and $W_2|_Y$ are equivalent whenever the projections $W_1|_X$ and $W_2|_X$ are equivalent.

Functional Dependencies – A Second Interpretation: Consider an XML data tree T' which is compatible with T and let ϕ be the unique homomorphism between them. T' *satisfies the XFD* $v : X \to Y$ if for any two **pre-images** W_1 and W_2 of $T(v)$ in T' the projections $W_1|_Y$ and $W_2|_Y$ are equivalent whenever the projections $W_1|_X$ and $W_2|_X$ are equivalent.

In order to distinguish between these two interpretations we shall speak of *functional dependencies of the first kind* (or *XFD*$_1$, for short) and *functional dependencies of the second kind* (or *XFD*$_2$, for short).

Reconsider the examples of functional dependencies discussed in Section 2. In the XML data tree in Fig. 1, we observed that He determines She within a pair. This gives rise to the XFD $v_{Pair} : X \to Y$, where X is the v_{Pair}-subgraph with the single leaf v_{He}, while Y is the v_{Pair}-subgraph with the single leaf v_{She}. The XML data tree in Fig. 1 satisfies this functional dependency both as an XFD$_1$ and as an XFD$_2$. In the XML data tree in Fig. 2, we noticed that for every course the set of pairs determines the rating. This leads to the functional dependency $v_{Course} : X \to Y$, where X is the v_{Course}-subgraph with the leaves v_{He} and v_{She}, while Y is the v_{Course}-subgraph with the single leaf v_{Rating}. The XML data tree in Fig. 1 satisfies this functional dependency as an XFD$_2$, but not as an XFD$_1$.

5 Interactions between XFDs

As for ordinary database relations, the integrity satisfied by an XML data tree are usually not independent. The notions of implication and derivability (with respect to some rule system $\mathcal{R}$) can be defined analogously to the notions in the relational database model, cf. [2]. A single constraint σ *follows* from a constraint set Σ if σ holds in every XML data tree which satisfies Σ. We also say that Σ *implies* σ. Of course, one will not inspect all possible XML data trees compatible with a given XML schema graph in order to decide implications of a given set of XFDs. Rather, we are interested in inference rules which help to decide this question. An *inference rule* is an expression $\frac{\Sigma'}{\sigma}\gamma$ where Σ' is a subset of Σ, and

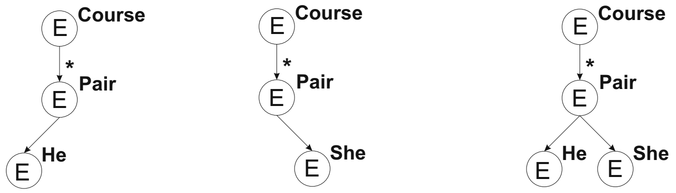


Fig. 12. The v_{Course}-subgraphs X, Y and $X \cup Y$ of the XML schema tree in Fig. 5.

γ states some condition on Σ' which has to be satisfied if we want to apply this rule. If Σ contains a subset Σ' satisfying the condition γ, then σ may be *derived* from Σ due to that inference rule. An inference rule is *sound* if Σ implies every constraint σ which may be derived from Σ due to that rule. The general problem is to find a rule system $\mathcal{R}$ which allows us to derive every constraint σ which is implied by a given set Σ within a fixed constraint class. Such a rule system is said to be *complete* for this class.

Let X, Y, Z be v-subgraphs of a given XML schema tree T, and let $X \cup Y$ denote the union of X and Y which is again a v-subgraph. The following rules may easily be obtained as generalizations of the well-known Armstrong rules for functional dependencies in the relational database model [5,6]:

$$\frac{}{v : X \to Y} \text{ Y is a } v\text{-subgraph of } X, \quad \frac{v : X \to Y}{v : X \to X \cup Y}, \quad \frac{v : X \to Y, v : Y \to Z}{v : X \to Z},$$

called reflexivity axiom, extension rule and transitivity rule. The reflexivity axiom and the transitivity rule are sound for both kinds of XFDs, that is, XFD_1s and XFD_2s. Surprisingly, however, the extension rule is only sound for XFD_1s, but not for XFD_2s as the following example shows. Consider the XML schema tree T in Fig. 5 and its v_{Course}-subgraphs X, Y and $X \cup Y$ shown in Fig. 12.

Further consider the XML data tree T' in Fig. 4. The pre-images of the total v_{course}-subgraph of the XML schema tree T are shown in Fig. 7, and their projections to the subgraphs X, Y and $X \cup Y$ in Fig. 13. The projections of the second and the third pre-image to X are equivalent, and so are the projections to Y. (Note, that the set of successors of a vertex is considered to be unordered. This is a usual convention in XML documents. For a discussion of this issue, we refer to Arenas/Libkin [4].) But the projections to $X \cup Y$ are not equivalent. Hence, the XFD_2 $v_{Course} : X \to Y$ holds, while the XFD_2 $v_{Course} : X \to X \cup Y$ is violated. This proves the extension rule to be wrong for XFD_2s in general.

It can be shown that for XFD_2s there is only a *restricted extension rule* applicable, namely

$$\frac{v : X \to Y}{v : X \to X \cup Y} \ X \text{ and } Y \text{ are reconcilable,}$$

where two v-subgraphs X, Y are called *reconcilable* if there are v-subgraphs X' of X and Y' of Y such that X' and Y' share no arc of frequency $*$ and such that

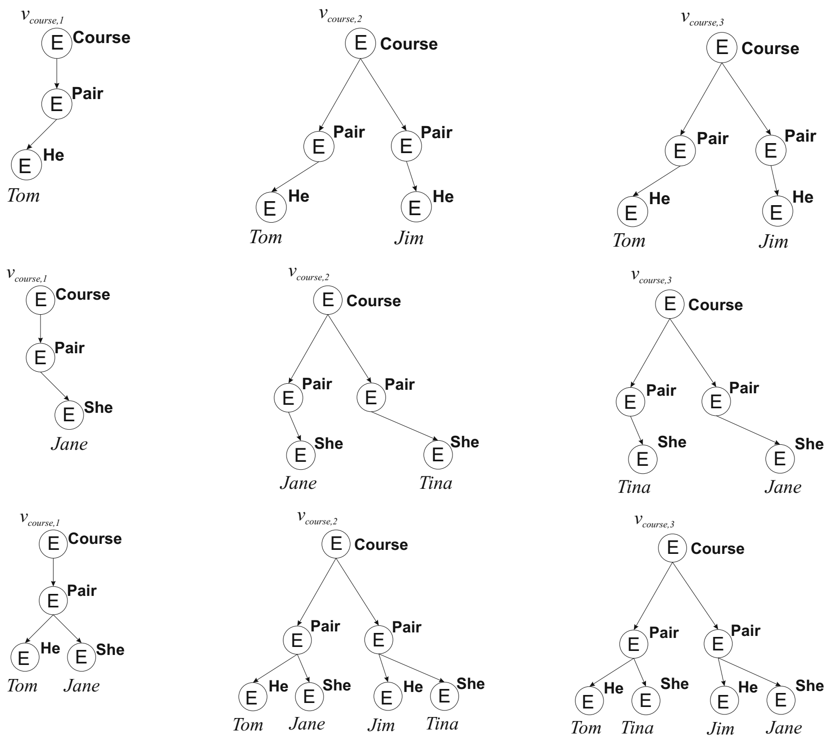


Fig. 13. The projections of the three pre-images of the total v_{Course}-subgraph of the XML schema tree T in Fig. 5 to the v_{Course}-subgraphs X, Y and $X \cup Y$.

$X \cup Y = X' \cup Y'$ holds. This means, that whenever X and Y share some arc (u, w) of frequency $*$, then X contains the total w-subtree of Y or Y contains the total w-subtree of X. In the example above this condition is violated as X and Y both contain the arc (v_{Course}, v_{Pair}), while their total v_{Pair}-subgraphs are obviously no subgraphs of one another, see Fig. 12.

In order to obtain a sound and complete rule system for XFD$_2$s defined with respect to a fixed vertex v of the XML schema tree T, we also need an additional axiom

$$\frac{}{v : X \to T(v)} \text{ all arcs of the directed path from } r_T \text{ to } v \text{ in } T \text{ have frequency 1,}$$

which points out that the total v-subgraph $T(v)$ of T has at most one pre-image in T' unless the directed path from r_T to v contains some arc of frequency $*$.

Finally, it should be mentioned that there are also interactions between XFD$_1$s and XFD$_2$s which give rise to the rule

$$\frac{\text{XFD}_1 \quad v : X \to Y}{\text{XFD}_2 \quad v : X \to Y}.$$

As the example in Section 2 shows, the converse of this rule is not sound in general. This observation points out that XFD$_1$s impose stronger conditions on an XML data tree than XFD$_2$s do. Conversely, however, this means that an XML data tree can well violate $v : X \to Y$ as an XFD$_1$, but satisfy $v : X \to Y$ as an XFD$_2$.

6 Conclusion

In this paper we presented an approach towards functional dependencies for XML based on homomorphisms between XML data trees and XML schema graphs. Our approach allows us to capture functional dependencies similar to those studied in [4,13], but also a new class of functional dependencies in XML data trees. Both kinds of XFDs are likely to occur in XML documents and should be considered in order to describe well-designed XML documents. Consequently, they should not be treated as controversial concepts, but as concepts which complement each other.

As mentioned in the introduction, there is an increasing number of applications using persistent XML documents, for example in molecular biology [19]. XML provides the capability of representing protein data or genetic data in a single, standardized data structure. Its simple syntax and the large variety of tools developed for handling XML documents are greatly expected to facilitate the management of molecular biology data. The success of XML raises hope that XML will soon serve as a first-class data model for storing and handling all kinds of complex data. For that purpose, however, a dependency theory similar to the relational database model has to be developed. Note that XFD$_2$s, studied within this paper, frequently appear in molecular biology data, e.g. in situations where the outcome of some analysis of genetic information does not depend on a single input factor, but on a set of factors.

We studied interactions of XFDs and, in particular demonstrated that the well-known extension rule is no longer valid for XFDs based on pre-images (that is, XFD$_2$s). Instead we presented a restricted extension rule which, together with some other rules, is powerful enough to describe implications of XFD$_2$s. Finally, we pointed out that XFDs based on maximal subcopies (that is, XFD$_1$s) always imply XFD$_2$s, but not the other way round. Hence there are XML documents violating an XFD$_1$, but satisfying the corresponding XFD$_2$. This raises the question for normal forms of XML data trees avoiding redundancy and update anomalies and ensuring well-designed XML documents. Arenas/Libkin [4] proposed such a normal form for XFDs based on maximal subcopies. Unfortunately, this normal form does not exclude redundancy due to XFDs based on pre-images. The same holds for other proposed normal forms e.g. [20] which, of course, have not been specially developed for XFD$_2$s. Nevertheless, we are confident that a slightly refined normal form can be defined which takes care of XFD$_2$s, too. This, however, is out of the scope of the present paper.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the web: From relations to semistructured data and XML.* Morgan Kaufmann, 2000.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases.* Addison-Wesley, 1995.
3. M. Arenas, W. Fan, and L. Libkin. What's hard about XML schema constraints. In *DEXA 2002*, pages 269–278, 2002.
4. M. Arenas and L. Libkin. A normal form for XML documents. In *PODS 2002*, pages 85–96. ACM, 2002.
5. W. W. Armstrong. Dependency structures of database relationship. *Inform. Process.*, 74:580–583, 1974.
6. W. W. Armstrong, Y. Nakamura, and P. Rudnicki. Armstrong's axioms. *J. Formalized Math.*, 14, 2000.
7. T. Bray, J. Paoli, and C. Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0*, W3C Recommendation, 2000.
8. P. Buneman, S. Davidson, W. Fan, and C. Hara. Reasoning about keys for XML. In *DBPL 2001*. Springer, 2001.
9. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39:473–487, 2002.
10. W. Fan and L. Libkin. On XML constraints in the presence of DTDs. In *PODS 2001*, pages 114–125. ACM, 2001.
11. W. Fan and J. Simeon. Integrity constraints for XML. In *PODS 2000*, pages 23–34. ACM, 2000.
12. D. Jungnickel. *Graphs, networks and algorithms.* Springer, 1999.
13. M. Lee, T. Ling, and W. Low. Designing functional dependencies for XML. In *EDBT 2002*, pages 124–141. Springer, 2002.
14. T. Ling, M. Lee, and G. Dobbie. Applications of ORA-SS. In *IIWAS 2001*, pages 17–28. Springer, 2001.
15. B. Thalheim. *Entity-relationship modeling.* Springer, Berlin, 2000.
16. H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, editors. *XML Schema Part 1: Structures*, W3C Recommendation, 2001.
17. V. Vianu. A web odyssey: From Codd to XML. In *PODS 2001*. ACM, 2001.
18. G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Trans. Database Systems*, 17:32–64, 1992.
19. R. K. Wong and W. Shui. Utilizing multiple bioinformatics information sources: An XML database approach. In *BIBE 2001*, pages 73–80. Springer, 2001.
20. X. Wu, T. Ling, S. Lee, M. Lee, and G. Dobbie. NF-SS: A normal form for semistructured schema. In *ER 2001 Workshops*, pages 292–305. Springer, 2001.

Appendix

```
<Root>
 <Course Date=”Feb 1”>
   <Pair>
     <He>Tom</He>
     <She>Jane</She>
   </Pair>
   <Ranking>A-</Ranking>
 </Course>
 <Course Date=”Feb 2”>
   <Pair>
     <He>Tom</He>
     <She>Jane</She>
   </Pair>
   <Pair>
     <He>Jim</He>
     <She>Tina</She>
   </Pair>
   <Ranking>B-</Ranking>
 </Course>
 <Course Date=”Feb 5”>
   <Pair>
     <He>Tom</He>
     <She>Jane</She>
   </Pair>
   <Pair>
     <He>Jim</He>
     <She>Tina</She>
   </Pair>
   <Ranking>B-</Ranking>
 </Course>
</Root>
```

Fig. 14. An XML document corresponding to the XML data tree in Fig. 1.

Towards Collaborative Video Authoring[*]

Boris Novikov and Oleg Proskurnin

University of St.Petersburg, Russia
{borisnov,olegpro}@acm.org

Abstract. It's a long time since video post production became digital, yet the issue of collaborative video authoring has not been seriously investigated so far. In this paper we tackle this problem from a transactional point of view, that allows us to ensure consistent exchange and sharing of information between authors by taking into account application semantics. Our main contribution is the development of a new data model called concurrent video, especially intended for cooperative authoring environments. We demonstrate that the presented model provides efficient means of organizing and manipulating video data, at the same time enabling direct use of merging mechanisms, which constitute a formal basis for collaborative scenarios. Moreover, since the proposed approach is mainly media-independent, we argue that the results of our work are applicable to other types of stream data as well.

1 Introduction

Today, computers are widely used to support the work of individuals. However, people tend to work in teams, whose members may be distributed over different locations. In addition, cooperation is inevitable in large-scale development and joint efforts help to meet tight deadlines in shorter projects. Thus, computers are required to support team activities just as well as individual tasks.

In case of video authoring collaboration often becomes an essential part of the process. This is especially true for the post production stage of movie-making when several people may be constantly involved in editing, adding visual effects and assembling a set of clips coming from common footage. Undoubtedly, appearance of adequate tools for multi-user video authoring is going to encourage closer cooperation among editors and hence enhance the quality of work that is carried out in both entertaining and broadcasting industries [9].

Since the goal of team work on a shared resource is to increase development productivity, cooperating users should not be disturbed with various difficulties that may arise from parallel actions of other participants. Therefore, ensuring consistency of shared data in the presence of concurrent actions becomes the key aspect in collaborative applications design. Actually, the latter happens to be the classical problem of transaction management, except that the traditional paradigm of competition for resources has to be replaced with an appropriate cooperation strategy.

[*] This work is partially supported by the Russian Foundation for Basic Research under grant No. 01-01-00935.

Such technique was presented in the cooperative activity model CoAct [15], which provides basic transactional support for interactive multi-user environments. In CoAct, each user operates in its own workspace that contains private versions of shared objects. User's actions are modeled via an activity history, which can be treated as a sequence of operations representing an individual working process. To achieve cooperation, co-workers can explicitly incorporate the results of their actions into the common workspace, which reflects the current state of collaborative effort. This incorporation can be viewed as semantically correct merging of private activity histories [20], guided by compatibility relations of constituent operations and involving the responsible user in case of a conflict. Thus, user transactions are executed concurrently in the context of a single cooperative transaction, enabling joint work and ensuring consistency of shared data.

In this paper we utilize formal aspects of the CoAct framework, which functionality corresponds to the needs of collaborative authoring environments [16, 1], in the field of cooperative video editing. For this purpose we develop the concurrent video data model, which provides an efficient tree-based abstraction for the underlying media stream and defines basic editing operations to support the authoring process. Furthermore, the model enables adequate maintenance of activity histories, facilitating utilization of the CoAct merging algorithm in the considered applications.

The rest of the paper is organized as follows. After giving a brief overview of related work, in section 3 we introduce the concurrent video data model. Next, in section 4, we discuss the transactional aspect of our proposal. Finally, we present conclusions and further work.

2 Related Work

In this section we first describe basic concepts underlying non-linear video editing and then briefly review previous approaches to cooperative multimedia authoring. Lastly, we summarize current research on video modeling.

2.1 Video Authoring

During the last decade video post-production techniques have moved from analog tape-to-tape editing to the world of digital non-linear editing systems. Computers brought enormous advantages to the video industry, leaving behind the drawbacks of work with sequential storage media.

Though there are various approaches to digital multimedia authoring [6], in this paper we will refer to well-known timeline-based systems to understand the basics of non-linear editing. We will keep in mind the functionality offered by representative commercial tools such as Adobe Premiere [3] and Ulead MediaStudio Pro [17] to identify the most general concepts underlying the video authoring process.

In applications we consider, the movie is created by altering and assembling source clips that are treated as independent units of raw material. These clips along with other graphical and audio objects are placed on the timeline, which naturally represents the time flow [18]. After the inclusion of necessary objects, they can be edited, special effects and transitions can be applied and the resulting material can be arranged into a final video production. Thus, the authors are likely to operate on clips as a whole, rather than on particular video frames. In any case, such editing implies just referencing the original files, which are not actually modified, and hence a special process called rendering is required to preview or export the produced movie.

To the best of our knowledge, none of the existing commercial applications supports concurrent video authoring and the related research works are also very few.

In [5] and [19] cooperative editing of multimedia documents based on locking mechanisms is proposed, while in [24] the prospects of exploiting operational transformations in a replicated multimedia authoring environment are investigated. However, these approaches do not consider stream data modeling issues and do not provide consistency guarantees in a transactional sense.

2.2 Video Modeling

There exists a wealth of research proposals covering various aspects of video modeling, such as annotation, content-based access and segmentation. Taking into account the course of our work, we will focus on those models that consider composition of structured multimedia data.

A set of basic operations for temporal and spatial composition of video segments is defined in video algebra [23]. These operations allow users to create multi-window presentations by constructing possibly nested algebraic expressions. A hierarchy of annotated expressions forms the algebraic video data model, providing a logical representation to the underlying raw material and enabling content-based access.

OVID database system [14] introduces video-objects as an abstraction over sets of video frames. Each object has a unique identifier and an appropriate set of descriptive attributes, which can be inherited by newly composed elements on the basis of the interval inclusion relationship.

AVIS system [2] presents a tree-based structure for modeling video and its contents. Though this abstraction is mostly designed for annotation and query processing, adequate updating mechanisms are also supported. The latter makes this work in some respects the closest to our investigations.

A recent proposal [8] to a certain extent summarizes previous approaches, constructing a framework for segmentation, annotation, querying and composition of video data. In this model, composition operators are embedded into the query language and propagate the logical structure of the source videos to the resulting ones.

However, editing operations in the above systems, as well as those in some other related works [7, 10], will unlikely be sufficient for multimedia authoring environments. The problem lies in the fact that most mentioned models do not provide explicit support for such essential editing actions as insertion and deletion of video segments or frame-level manipulations.

Actually, a model called stream algebra [13] does provide a formal basis for operating on diverse kinds of streams in the context of the exploratory data analysis. It defines some useful editing operations, but it does not consider the logical structure of the underlying media and therefore can hardly be exploited in video authoring applications without any additional abstractions.

3 Concurrent Video Model

In this section we first emphasize some important issues related to modeling video data in collaborative authoring environments and then present our solution to these problems – the concurrent video model.

3.1 Basic Requirements

Generally, the main goal of our work is to develop a data model that would be suitable for the needs of video authoring techniques (1) and that would enable execution of cooperative activities (2).

More precisely, the first requirement implies the creation of an adequate internal representation of the media stream, that would capture the nature of the editing process and would provide efficient support for manipulating underlying video. Traditionally, such task assumes the development of an appropriate abstraction that merely references raw material, being separate from it, and also contains various additional information, i.e. meta-data. In this way, instead of working with vast amounts of multimedia structures, references to videos are manipulated, supporting efficient updating mechanisms and promising some other benefits [10].

At the same time the second requirement adds a transactional aspect to our investigations. Since in the CoAct framework the compatibility property is used as a basis for merging activity histories and, similar to conventional concurrency control [22], a semantic conflict test is defined for each possible pair of operation invocations, the model should provide editing operations with good commutativity properties, thus enabling high concurrency among co-workers' actions.

A more serious problem here is maintaining activity histories, which play the central role in merging facilities of CoAct. The point is that the presence of inverse operations, such as insertion and deletion of video segments, may lead to fictitious conflicts and accumulation of large amount of redundant data in these histories, as demonstrated later in section 4. Mentioned issues are not addressed in CoAct and thus certain mechanisms should be provided to enable correct use of the existing merging algorithm in the discussed applications.

Concurrent video satisfies both of the above requirements in a uniform and efficient manner, presenting a single data structure for modeling underlying media and maintaining activity histories. According to the concepts of timeline-based authoring, the model proposes video segments of an arbitrary length as the basic units constituting the media stream. It provides a tree-based structure for referencing this raw material and supports high-level operations allowing the users to insert and delete, alter and temporally combine video clips to form the final production.

To enable cooperative efforts, operations on different segments are considered to be commutative, i.e. different users can work concurrently on different clips without any conflicts. Moreover, activity histories are stored within the same hierarchical structure that references video data, providing elegant support for merging mechanisms and eliminating all problems related to the history maintenance.

In what follows we give the definition of the concurrent video data model, while the transactional aspect of our work is discussed later in section 4.

3.2 Video Segments

We introduce the notion of a video segment to represent an abstraction over independent units of video data, which are used as building blocks within the authoring process. Basically, each video segment references a contiguous part of raw material via a frame sequence and in addition has its own set of attribute-value pairs which describe the proper interpretation of the underlying media at the presentation and rendering level. Frame sequences reflect the stream-like nature of video data, while attributes support implementation of non-destructive editing operations.

Following are the definitions:

Definition 1 (Frame Sequence). A frame sequence is a finite non-empty sequence $(f_1, \ldots, f_N)$ of video frames f_i, referring to a contiguous block of video data. N is called the length of a video sequence F and is denoted by $Length(F)$.

Definition 2 (Video Segment). A video segment is a pair (F, A), where F is a frame sequence and A is a possibly empty set of attribute-value pairs $\{a_i{:}v_i\}$, storing various additional information about the containing segment.

To refer to different elements of the above entities and other objects introduced later we will use dot notation. For example, $S.F$ will identify the frame sequence of a given video segment S.

In general, the presented definitions are similar to those that can be found in previous research works, for example see [10], and our notion of a video segment is actually very close to the notion of a video object in [14]. However, we intentionally avoid timing specifications within frame sequences and do not interpret segment's attributes as mere annotations to the underlying data.

The reason why we do not explicitly consider the temporal aspect in the definition of a frame sequence is the assumption of constancy of the frame rate within a particular block of raw video, to a portion of which the sequence is supposed to refer. The corresponding frame rate can be easily indicated aside among the attributes of the given video segment, for example as $\{FrameRate{:}30\}$, thus allowing users to change the play speed of the selected clips in an efficient way and without modifying the source material.

In a similar manner, attributes can be used to specify the spatial resolution or the color mode desired in the final production. Moreover, modeling of special effects and transitions, which are typically applied to video clips, is also possible in this way, as it will be described in more detail in section 3.4.

Below, for convenience of the subsequent discussion, we define two trivial operations on frame sequences:

Definition 3 (Concatenation). For any two given frame sequences $F = (f_1, \ldots, f_N)$ and $G = (g_1, \ldots, g_M)$ their concatenation is a new frame sequence $(f_1, \ldots, f_N, g_1, \ldots, g_M)$, denoted by $Concat(F, G)$.

Definition 4 (Extraction). For any given frame sequence $F = (f_1, \ldots, f_N)$ of length N, an extraction within the range $[k, m]$, where $k \geq 1 \wedge m \leq N \wedge k \leq m$, is a new frame sequence $(f_k, f_{k+1}, \ldots, f_m)$, denoted by $Extract(F, [k, m])$.

Advanced versions of these operations can be found in [8].

3.3 Video Activity Tree

At this point we introduce the core part of the concurrent video data model, defining a hierarchical structure that arranges video segments in a single media stream and at the same time enables efficient maintenance of activity histories. Basically, each video activity tree represents an abstraction over a sequence of clips which can be edited and temporally combined by cooperating users to form the final production.

The proposed tree is a properly organized collection of nodes which serve different purposes depending on their position within the hierarchy. Particularly, certain leaf nodes, called valid, are associated with video segments constituting the media stream, other leaves, called dead, stand for previously deleted segments and intermediate nodes are mainly intended for holding related parts of the activity history. Actually, there exists a one-to-one correspondence between valid nodes of the tree and video clips visible to the users, and moreover, the total order imposed on leaf nodes by the tree structure reflects the order in which mentioned clips appear on the timeline. Additionally, since we are concerned with collaborative work, nodes as well as activity history elements are marked whether they are private or shared, indicating what part of the tree and the history is present in the common database and what part exists only in the considered local workspace.

The following is a basic illustration of the above concepts:

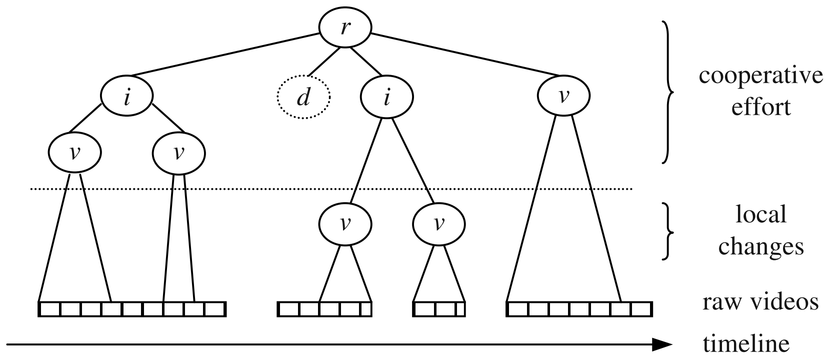


Fig. 1. A video activity tree in a local workspace. Characters *v*, *d*, *i* and *r* denote valid, dead, intermediate nodes and the root node of the tree correspondingly

One of the most important aspects of our data model is the assignment of unique identifiers to valid nodes of the video activity tree. The point is that utilization of these unique values for referring to video clips allows us to develop efficient editing operations with state-independent commutativity relations. This is achieved due to the fact that the considered identifiers do not depend on the actual positions of respective video segments on a timeline, which are also unique, but often change as the users insert new material into the media stream or remove unnecessary parts from it.

Generally, our approach is noticeably different from previous tree-based models, such as those presented in [2] and [12], which are mainly devoted to annotation and content-based querying of video databases rather than to video composition and cooperative work.

Now, let's move on to the formal definition of a video activity tree:

Definition 5 (Video Activity Tree). A video activity tree *T* is a tuple (*Root*, *Nodes*, *Valid*, *Dead*), where *Root* is the root node of *T*, *Nodes* is a set of intermediate nodes of *T*, and *Valid* and *Dead* are disjoint sets of so-called valid and dead nodes, which are all leaves of *T*, with and without associated video segments correspondingly.

For any given node $N \in \{Root\} \cup Nodes \cup Valid \cup Dead$ its parent node is denoted by *Parent*(*N*) and a set of its child nodes is denoted by *Children*(*N*).

Additionally, there exists a total order $<_t$ defined over a set of leaf nodes of *T*, that corresponds to the order in which associated video segments are located (or were located for dead nodes) on the timeline, i.e.: $\forall\, N, M \in Valid \cup Dead$: $N <_t M \vee M <_t N$.

For the clarity of further discussion we will not dwell on the implementation of the leaves order and child-parent relationships, which exist in a video activity tree. However, before presenting the internal structure of particular nodes, we will introduce the notion of an operation instance that is crucial for understanding activity histories:

Definition 6 (Operation Instance). An operation instance is a tuple (*Status*, *OID*, *Name*, *Input*, *Output*), where $Status \in \{private, shared\}$ indicates whether this instance is present only in the current workspace or not, *OID* is a unique identifier of this instance and *Name* is the name of the corresponding operation, whose input and output parameters are reflected in *Input* and *Output* sets.

Intuitively, an operation instance can be treated as a record establishing the fact of execution of a certain editing operation by some user. Such records act as elementary entities for modeling the authoring process and building activity histories. Furthermore, they have unique identifiers for tracing identical instances during merging.

In the concurrent video model, operation instances are stored in the nodes of a video activity tree along with other relevant information in the following manner:

Definition 7 (Valid Node). A valid node *V* of a video activity tree is a tuple (*NID*, *Segment*, *History*), where *NID* is a unique node identifier of *V*, *Segment* is a video segment associated with *V* and *History* is an ordered set of operation instances related to the given node *V*.

Definition 8 (Intermediate Node and Dead Node). An intermediate node as well as a dead node of a video activity tree is merely a tuple (*History*), where *History* is intended for the same purpose as its counterpart in valid nodes.

From the above definitions it should be clear that in the presented model raw video data is referenced only by those video segments which are contained in valid nodes of the video activity tree. One of the key aspects is that these segments can be addressed by means of the dedicated identifiers, which are unique for each valid node. Such values can be generated, for example, by combining in a single entity a unique name of the current user and a trivial counter of identifiers that were assigned to particular video clips during the overall work of that user.

In general, all data structures defined above are managed by editing operations provided by the concurrent video model and partly by cooperation primitives which are responsible for information exchange in the collaborative environment and which are in fact supposed to update status of the operation instances only.

3.4 Editing Operations

Evidently, cooperating users involved in the authoring process are not supposed to know anything about the internals of the concurrent video model. The only related thing that they see is the timeline and the most common thing that they are likely to do is altering and assembling the selected video clips.

To support such kind of work we introduce appropriate editing operations, which can be easily utilized for modeling user's actions within a particular workspace. These operations adequately modify the video activity tree, preserving the correspondence between ordered valid nodes and video clips on a timeline and at the same time efficiently maintaining the activity history.

First of all, we define an initialization routine that should be used to form an initial state of the cooperative activity in the common database.

Definition 9 (Initialization Algorithm). The initialization algorithm takes a sequence of N video clips, denoted by a collection of N video segments (vs_1, …, vs_N) as input, forms a corresponding video activity tree with an empty history and yields no output:

1. Construct N valid nodes V_i: $\forall\, i,\ 1 \le i \le N$: $V_i := (NewID(), vs_i, \varnothing)$,
   where $NewID()$ denotes a function generating a new unique identifier.
2. Construct a video activity tree T: $T := (Root, \varnothing, \{V_i \mid 1 \le i \le N\}, \varnothing)$, such that:
   $\forall\, V_i$: $Parent(V_i) = Root \wedge \forall\, i, j$: $1 \le i < j \le N \Leftrightarrow V_i <_t V_j$.

The initialization algorithm is not considered as an operation that should be reflected in the activity history, it merely constructs an initial version of the video activity tree from the specified ordered set of video clips, which then can be manipulated by means of other operations to form the final production.

Typically, throughout the editing process authors may wish to include some new clips into their project as well as remove existing ones. This can be achieved with the help of insertion and deletion operations, which allow users to insert a particular video fragment into the specified position on a timeline or delete a selected video clip, accordingly shifting the subsequent material in both cases.

Definition 10 (Insertion Algorithm). The insertion algorithm takes as input a new video fragment, denoted by the segment vs, and its desired location on the timeline, specified by a destination point pos within the frame sequence of a clip with given identifier id. The algorithm splits the affected node of the video activity tree T, updates its history and yields a set of newly created clip identifiers $\{V_i.NID\}$ as output:

1. Find a node $V \in T.Valid$: $V.NID = id$, report failure if such node does not exist.
2. Construct an intermediate node V': $V' := (V.History)$.
3. *If* $pos > 0 \wedge pos < len,$ where $len = Length(V.Segment.F)$, construct valid nodes V_i:
   $V_1 = (NewID(), (Extract(V.Segment.F, [1, pos]), V.Segment.A), \varnothing)$,
   $V_2 = (NewID(), vs, \varnothing)$,
   $V_3 = (NewID(), (Extract(V.Segment.F, [pos + 1, len]), V.Segment.A), \varnothing)$,
   Else construct only two of the above nodes having non-empty frame sequences.
4. Adjust T: $T := (Root, T.Nodes \cup \{V'\}, T.Valid \cup \{V_i\} \setminus \{V\}, T.Dead)$, such that:
   $Parent(V') = Parent(V) \wedge \forall\, V_i$: $Parent(V_i) = V' \wedge \forall\, i, j$: $i < j \Leftrightarrow V_i <_t V_j$.
5. Append instance $(private, NewID(), Insert, \{vs, id, pos\}, \{V_i.NID\})$ to $V'.History$.

Since the insertion algorithm is based on the node splitting technique, the affected valid node is replaced with a subtree whose leaves correspond to the resulting combination of involved video clips, as illustrated in figure 2. Still, the actual behavior of this operation depends on whether the user inserts new material between two adjacent video segments or inside one of them. The former case is accomplished straightforward, while in the latter case the destination clip should be properly divided into two independent fragments which inherit their attributes from the common predecessor.

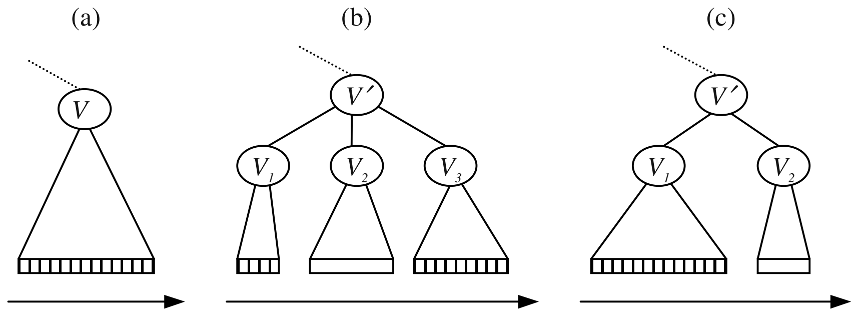


Fig. 2. Node splitting carried out during the insertion: (*a*) – an original valid node with the associated clip, (*b*) – the splitted node after an insertion inside the clip, (*c*) – the splitted node after an insertion to the end of the clip (insertion to the beginning of the clip is analogous)

Apparently, the discussed algorithm affects a bounded part of the video activity tree, operating on a single valid node and enabling commutativity of insertions executed within different video segments. Thus, users are able to add new material to different parts of the media stream concurrently.

At the same time, to delete previously inserted clips from the project authors are supposed to employ the next operation:

Definition 11 (Deletion Algorithm). The deletion algorithm takes as input an identifier *id* of the clip that has to be removed from the media stream. It correspondingly adjusts the video activity tree *T*, updates the history and yields no output:

1. *If* $\exists\ V \in T.Valid$: $V.NID = id$, *Then* find *P*: $P = Parent(V)$ and go to step 3.
2. *If* $\exists\ D \in T.Dead, \exists\ O \in D.History$: $O = (*, *, Delete, \{id\}, *)$, * denotes any value, *Then* report success, *Else* report failure.
3. *If* $\exists\ O \in P.History$: $O = (private, *, Insert, \{V.Segment, *, *\}, *) \wedge$
   $\forall\ N \in Children(P) \setminus \{V\}$: $N.History = \varnothing$,
   Then undo the effect of the insertion *O* and report success.
4. Construct a dead node V': $V' := (\{O \in V.History \mid O.Status = Shared\})$.
5. Adjust *T*: $T := (Root, T.Nodes, T.Valid \setminus \{V\}, T.Dead \cup \{V'\})$, such that:
   $Parent(V') = Parent(V) \wedge \forall\ N \in T.Valid \cup T.Dead$: $V' <_t N \Leftrightarrow V <_t N$.
6. Append an operation instance $(private, NewID(), Delete, \{id\}, \varnothing)$ to $V'.History$.

The presented deletion algorithm has several important features. First, it is designed in a way to support commutativity between any two removals, even those of the same clips, as specified in step 2. Second, it acts as an inverse for some insertions, that are not shared by other users and are not affected by subsequent operations, as checked in step 3. Evidently, such compensation can be performed by undoing the effect of node splitting, which may involve concatenation of the previously separated frame sequences, and by removing the corresponding insertion record from the history. Lastly, the discussed algorithm eliminates from the dead node those operation instances which are not shared and hence become irrelevant after the clip removal.

In general, the proposed operations provide efficient means of concurrent temporal composition of video segments within a single media stream. However, they do not provide adequate support for moving video clips along the timeline, since a pair of appropriate deletion and insertion methods will conflict with parallel editing of the moved clip. That's why we additionally develop the following algorithm:

Definition 12 (Moving Algorithm). The moving algorithm takes as input an identifier *mid* of the clip that has to be moved and its new location specified by *pos* and *id* parameters, which are similar to their counterparts in the insertion algorithm. The identifier of the moved clip is kept intact, the activity history is updated accordingly and a set of newly created node identifiers $\{V_i.NID \mid V_i.NID \neq mid\}$ is returned as output:

1. Find a node $M \in T.Valid$: $M.NID = mid$, fail if such node does not exist.
2. Find a node $S \in T.Valid$: $S.NID = id$, report failure if such node does not exist.
3. Apply steps 4 and 5 of the deletion algorithm to the node *M* to form a dead node *D*.
4. Apply steps 2 - 4 of the insertion algorithm to the nodes *S* (splitted) and *M* (moved) in such a way that: $\exists\, i$: $V_i \in Children(S) \wedge V_i = (mid, M.Segment, \varnothing)$.
5. Append an instance $(private, NewID(), Move, \{mid, id, pos\}, \{V_i.NID\})$ to the histories *D.History* and *S.History*.

In contrast with previous algorithms the one presented above alters two valid nodes of the video activity tree simultaneously and adds the same operation instance to the histories in both of them. But essentially, it is just a slightly modified sequence of deletion and insertion, which turns out to be commutative with the next operation:

Definition 13 (Editing Algorithm). The editing algorithm takes as input an identifier *id* of the clip that has to be modified and a new video segment *vs* which represents the results of the modification. The algorithm simply replaces an old video segment with the new one, updating the history and yielding no output:

1. Find a node $V \in T.Valid$: $V.NID = id$, report failure if such node does not exist.
2. Modify *V*: $V := (V.NID, vs, \{O \in V.History \mid O.Status = Shared\})$.
3. Append an operation instance $(private, NewID(), Edit, \{id, vs\}, \varnothing)$ to *V.History*.

Obviously, this algorithm is a rough approximation to the actual modifications, such as image-level manipulations, which can occur within a video clip. Nevertheless, we leave the development of intra-segment editing operations to further research, emphasizing that even the presented methods ensure a high level of concurrency.

Moreover, due to existence of the last algorithm we can model transitions and special effects by means of certain attribute-value pairs, which may, for example, describe a sequence of filters applied to the corresponding video clip or indicate a type of transition to the next clip. The values of these attributes may be changed at any time without modifying the source material and after all they can be exploited during the rendering stage for constructing the desired video production.

4 Collaborative Authoring

In this section we present a general description of how cooperation can be achieved in a multi-user video authoring environment by means of the concurrent video data model. This part of our investigations is entirely based on the CoAct transactional concepts, which can be found in [11, 20] and for brevity are not stated here.

At first, we provide an efficient method of extracting correct subhistories from a set of operation instances scattered over the video activity tree, thus enabling selection of consistent units of work that can be subject of information exchange. After that, we look in more detail at the history merging mechanism, pointing out some related problems and examining how they are solved in the concurrent video model.

4.1 Consistent Units of Work

In the CoAct framework two distinct types of commutativity, forward and backward, are exploited for manipulating activity histories [11, 21].

Forward commutativity was already mentioned in section 3.4 and will be thoroughly discussed a bit later, as exactly this kind of relation is used for detecting conflicts between operations coming from different histories during merging.

Backward commutativity serves another purpose and is discussed below, as it is intended for determining dependencies between user's actions within a single history. The point is that the behavior of editing actions is typically influenced by some of the previously executed operations, for example, any modifications of a particular video clip may depend on the preceding insertion of this clip into the media stream. Hence, no operation instance can be exchanged between workspaces without its relevant predecessors. And it is backward commutativity that allows us to identify so-called closed subhistories [11] which have no external dependencies and thus represent consistent units of work. Evidently, such subhistories can be exchanged between cooperating users separately from each other.

Turning back to the concurrent video data model, we aim at the development of a technique for constructing closed subhistories out of a given video activity tree. This implies selecting required operation instances from the nodes of the tree and defining a proper total execution order over these instances. The latter brings an additional complication to our discussion since in contrast to the CoAct framework we do not maintain activity histories as totally ordered sets of constituting elements and thus, the sequential nature of the histories should be somehow recovered at the stage of information exchange to enable cooperation.

We start with the definition of the following partial order, which naturally extends nodes child-parent relationships present in a video activity tree to operation instances:

Definition 14 (Hierarchical Order). For any two operation instances A and B, contained in a video activity tree T, we consider that $A <_H B$ iff:

$\exists N \in T.Nodes \cup T.Valid \cup T.Dead$: A precedes B in the ordered set $N.History$ $\vee$

$\exists P$: $P = Parent(N)$, A is the last element in $P.History$ and $B \in N.History$.

Actually, this partial order reflects all relevant dependencies which exist between non-commuting operations in a particular activity tree. In fact, it corresponds to the restricted (comparing to [11]) backward commutativity relation, which ensures that the execution order of any two successive commuting operations from the given history can be exchanged without affecting the state of a video tree. This means that the last definition specifies a basis for extracting those operation instances from an activity tree which belong to the same closed subhistory.

Moreover, determination of an appropriate total execution order within the selected subhistory becomes straightforward according to the following lemma, which provides a way of constructing equivalent legal histories [11] out of a given video tree:

Lemma 1. Any two total orders $<_U$ and $<_V$ that are defined over a set of operation instances H which are contained in the nodes of a given video activity tree T, such that $<_H \subset <_U \wedge <_H \subset <_V$, form two legal histories $(H, <_U)$ and $(H, <_V)$. The video activity trees resulting from applying these histories to the respective initial workspace state are identical.

Owing to the lack of space, we will omit the proof of the above statements as well as the proofs of some subsequent facts, paying more attention to the final results.

Briefly summarizing the previous discussion we want to emphasize that the selection of a closed subhistory from a video activity tree and its total ordering can be successfully implemented by means of the hierarchical order alone. And since this partial order is naturally reflected by our tree-based data model, we can easily provide an elegant and efficient algorithm that can be used for extracting consistent units of work from a given workspace:

Definition 15 (Subhistory Extraction Algorithm). The subhistory extraction algorithm denoted by *Subhistory* takes an operation instance O contained in a video activity tree T as input and yields as output an ordered list of operation instances from T representing a minimal closed subhistory [11] under $\{O\}$:

1. *If* $O = null \vee O$ is marked as already included in the subhistory *Then* return $\varnothing$.
2. Find a node $N \in T.Nodes \cup T.Dead \cup T.Valid$: $O \in N.History$.
3. *If* $O.Name = Move$ *Then* find a node $M \neq N$: $O \in M.History$ and go to step 6.
4. *If* $\exists A \in N.History \cup Parent(N).History$: $A <_H O \wedge \forall B <_H O$: $B <_H A$ *Then* $N_{prev} := A$ *Else* $N_{prev} := null$.
5. Mark O as already included in the subhistory and return $Subhistory(N_{prev}) \bullet O$.
6. Find N_{prev} and M_{prev} for nodes N and M like it is done in step 4.
7. Mark O as already included and return $Subhistory(N_{prev}) \bullet Subhistory(M_{prev}) \bullet O$.

Evidently, the extraction algorithm can be successfully used for constructing a minimal closed subhistory under a set of operation instances $\{O_1, \ldots, O_N\}$, with the help of mere lists concatenation (and under the assumption that inclusion markers are not reset between the function calls):

$List := Subhistory(O_1) \bullet \ldots \bullet Subhistory(O_N)$

Additionally, since any operation instance depends, possibly transitively, on all instances from the parents of its own node, we can supply a set of leaf nodes to the slightly modified version of the extraction algorithm for constructing the complete activity history of a given workspace. With such histories further exploitation of various aspects of the CoAct framework becomes possible.

4.2 Merging Activity Histories

In general, merging of activity histories in CoAct [20] is a semi-automatic process, which requires external control only in case of a conflict between users' actions. In what follows, we concentrate on utilization of the CoAct conflict detection technique, which does not suppose user interaction, within the concurrent video data model.

For this purpose we first present forward commutativity predicates of the concurrent video editing operations:

Table 1. Forward commutativity relation (symmetric)

Operations	*Edit*(*ID*, *VS*)	*Insert*(*VS*, *ID*, *POS*):*OUT*	*Delete*(*ID*)	*Move*(*MID*, *ID*, *POS*):*OUT*
Edit(*id*, *vs*)	$id \neq ID$			
Insert(*vs*, *id*, *pos*):*out*	$id \neq ID$	$id \neq ID$		
Delete(*id*)	$id \neq ID$	$id \neq ID$	*true*	
Move(*mid*, *id*, *pos*):*out*	$id \neq ID$	$mid \neq ID \wedge id \neq ID$	$mid \neq ID \wedge id \neq ID$	$\{mid, id\} \cap \{MID, ID\} = \varnothing$

Apparently, the predicates specified in the above table demonstrate that operations manipulating different video segments always commute. Moreover, the same clips can be deleted concurrently by several users, and video extracts moved along the timeline can be simultaneously modified by other participants. The latter is especially important since it allows cooperating authors to distinguish editing and ordering of particular clips as independent tasks. Thereby, the overall approach really offers a high level of concurrency to the co-workers, even though they are not encouraged to operate simultaneously on the same video clips.

Now, when proper commutativity relations are defined, they should be exploited to determine conflicts between users' activities. According to the history merging algorithm [20], two activity histories are basically considered to be compatible if their constituting operations pairwise commute. But generally, there may be certain situations in which this algorithm will lead to fictitious conflicts. For example, have a look at the following actions of Alice and Bob on some initial video activity tree:

Alice: [*Edit*(*ID*, *AliceVS*)]
Bob: [*Insert*(*NewVS*, *ID*, *0*):{*NewID*,..}] [*Edit*(*SomeID*, *BobVS*)] [*Delete*(*NewID*)]

Evidently, in his work Bob inserts a new video extract into the same clip which Alice edits and thus, a conflict is detected between these operations. However, later Bob removes the new segment from the media stream. Therefore, the mentioned conflict in fact stays only in the activity histories, but on the data level the results of such individual work are certainly compatible.

Fictitious conflicts, like one just described, take place since the deletion operation acts as an inverse for the insertion. This is the reason why we have designed the deletion algorithm of concurrent video in a way to rollback when possible the effect of the respective preceding insertion, as stated in definition 11.

Another noticeable aspect of history maintenance that we have introduced in our model is utilization of the masking concept [4]. Intuitively, a subsequent method masks a preceding one if the effects of the earlier operation turn out to be completely overwritten by the effects of the later one, thus, the former instance can be removed from the history without affecting the workspace state. In particular, the behavior of the deletion and editing algorithms, which remove preceding private instances from the processed node are based on the described concept.

As a consequence of these improvements in history maintenance, we become able to avoid certain fictitious conflicts and manage to reduce the amount of data kept within the nodes by removing masked operation instances. At the same time efficient implementation of the above features becomes possible due to storing of the activity history within a video tree.

5 Conclusions and Future Work

In this paper we have presented the concurrent video data model as a formal basis for collaborative video authoring environments.

The principal advantage of our approach lies in the fact that we have defined a single tree-based data structure for modeling both the underlying media stream and cooperative transactions. In spite of such duality, concurrent video undoubtedly satisfies basic requirements of authoring applications as well as the needs of multi-user environments in an elegant and efficient manner.

Moreover, the results of our work are general enough to be applicable to other types of stream data. For instance, by simple renaming of frame sequences and video segments into sample sequences and audio segments, the concurrent video model can be successfully transformed into concurrent audio that may be found suitable for cooperative editing of audio streams.

Finally, we consider incorporation of versioning support and development of intra-segment editing operations as probable directions of further research.

References

[1] Aberer, K., Klingemann, J., Tesch, T., Wasch, J., Neuhold, E.J.: Transaction models supporting cooperative work – the TransCoop experiences. In Proc. of the Int. Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, December 1996

[2] Adali, S., Candan, K.S., Chen, S., Erol, K., Subrahmanian, V.: The advanced video information system: Data structures and query processing. Multimedia Systems, Vol.4, pages 172–186, 1996

[3] Adobe, Adobe Premiere 6.5: Available at http://www.adobe.com/products/premiere, 2003

[4] Beaudouin-Lafon, M., Karsenty, A.: Transparency and awareness in a real-time groupware system. In Proc. of the 5th Annual ACM Symposium on User Interface Software and Technology, pages 171–180, Monteray, California, United States, 1992

[5] Borghoff, U.M., Teege, G.: Structure management in the collaborative multimedia editing system IRIS. In Proc. of the International Conference on Multi-Media Modeling (MMM'93), pages 159–173, Singapore, 1993

[6] Bulterman, D.C.A., Hardman, L.: Multimedia authoring tools: State of the art and research challenges. In Lecture Notes in Computer Science #1000, Springer-Verlag, 1995

[7] Duda, A., Keramane, C.: Structured temporal composition of multimedia data. In Proc. International Workshop on Multi-media Database Management Systems, Blue Mountain Lake, NY, August 1995

[8] Dumas, M., Lozano, R., Fauvet, M.-C., Martin, H., Scholl, P.-C.: A sequence-based object-oriented model for video databases. Multimedia Tools and Applications, Vol.18, Issue 3, pages 249–277, 2002
[9] Ghandeharizadeh, S., Kim, S.H.: Design of multi-user editing servers for continuous media. Multimedia Tools and Applications, Vol.11, Issue 1, pages 101–127, 2000
[10] Gibbs, S., Breiteneder, C., Tsichritzis, D.: Data modeling of time-based media. In Proc. of ACM SIGMOD International Conference on Management of Data, pages 91–102, Minneapolis, Minnesota, May 1994
[11] Klingemann, J., Tesch, T., Wasch, J.: Semantics-based transaction management for cooperative applications. In Proc. of the International Workshop on Advanced Transaction Models and Architectures (ATMA), pages 234–252, Goa, India, August - September 1996
[12] Li, J., Goralwalla, I., Ozsu, M., Szafron, D.: Modeling video temporal relationships in an* object database management system. In Proc. of IS&T/SPIE Int. Symposium on Electronic Imaging: Multimedia Computing and Networking, pages 80–91, San Jose, USA, 1997
[13] Mackay, W.E., Beaudouin-Lafon, M.: DIVA: exploratory data analysis with multimedia streams. Conference proceedings on Human factors in computing systems, pages 416–423, Los Angeles, California, United States, April 1998
[14] Oomoto, E., Tanaka, K.: OVID: Design and implementation of a video-object database system. IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.4, August 1993
[15] Rusinkiewicz, M., Klas, W., Tesch, T., Wasch, J., Muth, P.: Towards a cooperative transaction model - The cooperative activity model. In Proc. of the 21st International Conference on Very Large Databases, pages 194–205, Zurich, Switzerland, September 1995
[16] Tesch, T., Wasch, J.: Transaction support for cooperative hypermedia document authoring – a study on requirements. In Proc. of the 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems (EDRG-8), pages 31–42, Trondheim, Norway, August 1995
[17] Ulead, MediaStudio Pro 6.5: Available at http://www.ulead.com/msp/features.htm, 2003
[18] Wahl, T., Rothermel, K.: Representing time in multimedia systems. In Proc. of IEEE International Conference on Multimedia Computing and Systems, Boston, USA, May 1994
[19] Wang, K.: The design of extending individual multimedia authoring to cooperative multimedia authoring. In Proc. of the Fourth International Conference for Young Computer Scientists (ICYCS'95), Beijing, China, July 1995
[20] Wasch, J., Klas, W.: History merging as a mechanism for concurrency control in cooperative environments. In Proc. of RIDE-Interoperability of Nontraditional Database Systems, pages 76–85, New Orleans, USA, February 1996
[21] Weihl, W.E.: Commutativity-based concurrency control for abstract data types. IEEE Transactions on Computers, Vol.37, No.12, pages 1488–1505, December 1988
[22] Weikum, G.: Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, Vol.16, No.1, pages 132–180, March 1991
[23] Weiss, R., Duda, A., Gifford, D.: Composition and search with a video algebra. IEEE Multimedia, Vol.2, No.1, pages 12–25, 1995
[24] Xiao, B.: Collaborative multimedia authoring: scenarios and consistency maintenance. In Proc. of the Fourth International Workshop on Collaborative Editing Systems, New Orleans, Louisiana, United States, November 2002

Updatable XML Views[1]

Hanna Kozankiewicz[1], Jacek Leszczyłowski[1], and Kazimierz Subieta[1,2]

[1] Institute of Computer Science Polish Academy of Sciences, Warsaw, Poland
[2] Polish-Japanese Institute of Information Technology, Warsaw, Poland
{hanka,jacek,subieta}@ipipan.waw.pl

Abstract. XML views can be used in Web applications to resolve incompatibilities among heterogeneous XML sources. They allow to reduce the amount of data that a user has to deal with and to customize an XML source. We consider virtual updatable views for a query language addressing an XML native database. The novelty of the presented mechanism is inclusion of information about intents of updates into view definitions. This information takes the form of procedures that overload generic view updating operations. The mechanism requires integration of queries with imperative (procedural) statements and with procedures. This integration is possible within the Stack-Based Approach to query languages, which is based on the classical concepts of programming languages such as the environment stack and the paradigm of naming/scoping/binding. In the paper, we present the view mechanism describing its syntax, semantics and discussing examples illustrating its possible applications.

1 Introduction

XML has become of great interest for both internet and database community as it grows to be one of the most commonly used standards for data representation and exchange on the Web. Development of XML technologies shows a tendency to treat the Web as a semi-structured database consisting of multiple autonomous sites. In this context, views[2] can be considered as a virtual mapping of heterogeneous resources stored at remote sites to some unified business ontology. Views can offer many features like abstraction and generalization over data, transformation of data, access and merging data from multiple databases, etc. Assuming that views have full computational power, they fulfill the role that is typically assigned to mediators [Wied92], wrappers and adaptors. Such views addressing XML native databases can support indispensable qualities for many Web applications. The following general issues underlined in the database literature have their counterparts in applications of views to XML-based Web resources:

- *Customization, conceptualization and encapsulation.* Since users sharing the same XML data may have different needs, views enable them to see the same data differently - tailored to their interests and in the form suitable to their activity.

[1] Supported by the European Commission 5th Framework project ICONS (Intelligent Content Management System); no. IST-2001-32429.

[2] A view is a virtual image of data stored in a database/XML file. We deal with virtual views. Materialized views are a different research subject, almost entirely irrelevant to this paper.

- *Security, privacy and autonomy.* Views give the possibility to restrict user access to relevant parts of XML data.
- *Interoperability, heterogeneity, schema integration and legacy applications.* Views enable integration of distributed/heterogeneous data sources, allowing understanding and processing alien legacy or remote databases according to a common, unified schema.
- *Data independence and schema evolution.* Views enable users to change physical and logical data organization/schema without affecting already written applications.

These qualities are hard to satisfy goals, thus are challenging tasks for research and development. The literature concerning virtual views for object-oriented and object-relational databases has not yet presented a solution that would be satisfactory with respect to semantic clarity, implementability, computational universality, user friendliness and performance. This especially concerns updatable views that imply a difficult problem how to map updates addressing virtual data into updates of stored data.

Recently, views are the subject of research in the context of XML technologies [Abit00, KL02]. There have already been developed prototype view implementations [AAC+98, Lac01]. Papers on implemented views address also the area of semi-structured data [AGM+97, LAW99], closely related to XML data. These contributions show trends, but the subject still requires much more research and development.

In this paper, we present a new approach to virtual, updatable views for a query language addressing XML native databases. Updatability of views requires integration of update operations with query semantics. This excludes the classical approaches to query languages, such as object algebras and various forms of logic/calculi (in particular, an XML query algebra), because these frameworks do not deal with updating. We follow the Stack-Based Approach (SBA) to query languages [SBMS95, SKL95], which expresses the query semantics in terms of classical notions of programming languages, such as the environment stack, naming issues, scopes for names and binding names to run-time database/program entities. Our idea requires integration of queries with procedural statements and with procedures.

The novelty of our approach is introduction of information about intents of view updates into views' definitions. This is a revolutionary change in comparison to known approaches, which as a rule assume updates via some side effects of view definitions (e.g. by OIDs returned by a view invocation). Similarly to "instead of" triggers, the intents have a form of procedures that during run time perform view updating operations. However, in contrast to the "instead of" triggers, our approach addresses non-relational databases, it assumes overloading generic view updating operations by procedures rather than the event-action paradigm, it is much more general and it is optimisable. These features create possibilities, which have not been even considered in other approaches to updatable views, or have been considered as absolutely unfeasible. The idea is currently being implemented on top of an already implemented query language for XML native databases based on the DOM model.

The structure of the paper is as follows. The next section formalizes XML data structures, introduces main concepts of the Stack-Based Approach and presents its formalized OQL-like query language – SBQL. Section 3 presents our approach to updatable views. Section 4 includes examples manifesting the power of our method and its possible applications. Section 5 concludes.

2 Stack-Based Approach (SBA) for XML

XML is one of the most commonly used standards to represent data in the Web. More and more data are stored in XML or mapped to XML by means of wrappers from semi-structured documents, relational databases, object-relational and object databases. Popularity of XML is evident in commercial relational databases like Oracle, which includes integrated XML repository. This increasing amount of distributed information stored/presented in XML indicates a need for a method of their integration, filtering and searching. This is the role for a query language for XML and XML view mechanism. Below we present an XML query language that is based on SBA.

SBA is based on the assumption that query languages are a kind of programming languages. The approach is abstract and universal, which makes it relevant to a general object model, in particular to XML structures. SBA makes it possible to precisely define the semantics of query languages, their relationships with object-oriented concepts, with imperative programming constructs and with programming abstractions including procedures, functional procedures, views, modules, classes, methods, etc.

2.1 XML Object Store Model

XML data form a tree structure with nested tags representing a hierarchy of nested objects. XML tags can also have attributes, which formally can be treated similarly to nested XML documents. We extend the classical XML store model by introducing relationships (links) between objects. We present a formal model of XML document store, adapting it to the SBA terms below.

In SBA each object has the following components:

- Internal identifier (OID); identifiers cannot be directly written in queries and are not printable. Let I be a set of such internal identifiers.
- External name (introduced by a designer, programmer or user) used to access the object from a program. Let N be a set of such external names.
- Content that can be a value, a link, or a set of objects.

Let V be a set of atomic values, e.g. numbers, strings, blobs. Atomic values also include also codes of procedures, functions, methods, views and other procedural entities. Formally, objects are modeled as triples defined below, where $i, i_1, i_2 \in I, n \in N$ and $v \in V$:

- Atomic object *as* $<i, n, v>$.
- Link object *as* $<i_1, n, i_2>$.
- Compound object *as* $<i, n, S>$, where S denotes a set of objects.

The definition is recursive and allows creating compound objects isomorphic to XML documents with an arbitrary number of hierarchy levels. Relationships (associations) are modeled through link objects. In order to model collections, SBA does not impose the uniqueness of external names at any level of data hierarchy, like in XML. Although XML objects have no explicit identifiers (references) on the level of XML text files, they must appear in any form of parsed XML, e.g. in the DOM model. Some form of XML object identification is also present in XPath and Xlink utilities. In the above definition we do not determine (and are not interested in) a particular method of internal identification of XML documents.

In SBA, objects populate an *object store*, which is formed by:

- The structure of objects, subobjects, etc.
- OIDs of root objects which are starting points for querying.
- Constraints (e.g. uniqueness of OIDs, referential integrities, etc.).

Example Data Store. We present an example XML object store in Fig. 1. It contains a fragment of a database describing a company. The company has two employees; one of them being a chief of the IT department.

```
<Company>
  <Department id=„1”>
    <dNo>1</dNo>
    <dName>Computer Science</dName>
    <loc>Elms St. 21</loc>
    <loc>Wall St. 11</loc>
    <employs pointer=„true”>2</employs>
    <employs pointer=„true”>3</employs>
    <boss pointer=„true”>3</boss>
  </Department>

  <Employee id=„2” sex=„male”>
    <eNo>123</eNo>
    <name>John Smith</name>
    <job>designer</job>
    <sal>2345</sal>
    <rating>5.7</rating>
    <works_in pointer=„true”>1</works_in>
  </Employee>

  <Employee id=„3”>
    <eNo>456</eNo>
    <name>George Cooper</name>
    <job>consultant</job>
    <sal>7000</sal>
    <rating>3.1</rating>
    <works_in pointer=„true”>1</works_in>
    <manages>1</manages>
  </Employee>
</Company>
```

Objects:

<i_1, Company, {
  <i_2, Department, { <i_3, attr_id, 1>,
    <i_4, dNo,1>,
    <i_5, dName, „IT”>,
    <i_6, loc, „Elms St. 21”>,
    <i_7, loc, „Wall St. 11”>,
    <i_8, employs, i_{11}>,
    <i_9, employs, i_{20}>,
    <i_{10}, boss, i_{20}>}>,

  <i_{11}, Employee, { <i_{12}, attr_id, 2>,
    <i_{13}, attr_sex, „male”>
    <i_{14}, eNo, 123>,
    <i_{15}, name, „JohnSmith”>,
    <i_{16}, job, „designer”>,
    <i_{17}, sal, 2345>,
    <i_{18}, rating, 5.7>,
    <i_{19}, works_in, i_2> }>,

  <i_{20}, Employee, { <i_{21}, attr_id, 3>,
    <i_{22}, eNo, 456>,
    <i_{23}, name, „George Cooper”>,
    <i_{24}, job, „consultant”>,<i_{25}, sal, 7000>,
    <i_{26}, rating, 3.1>, <i_{27}, works_in, i_2>,
    <i_{28}, manages, i_2> }>
 }>

Roots: { i_1, i_2, i_{11}, i_{20}}

Fig. 1. The object store

Note that there is no problem to map XML into the presented object store - it is isomorphic to XML. We treat attributes of an object as its internal objects and we distinguish them from regular internal objects by name prefix “attr_”. We identify link objects in XML by the attribute *pointer* set to true; for short, we do not present explicitly attr_pointer objects, but only parsed version in the SBA convention. In our example roots for the store refer to the *Company* object and to *Department* and *Employee* sub-objects. This is an arbitrary choice. Alternatively, we could assume that roots are {i_2, i_{11}, i_{20}} or there is a single root {i_1}. Our assumption means that queries can start from *Company*, *Department* and *Employee*.

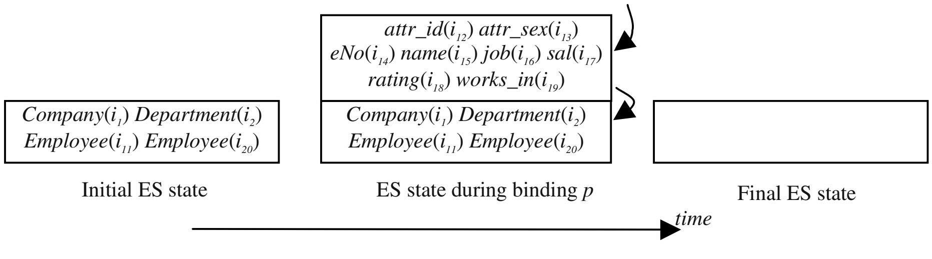
2.2 Environment Stack (ES)

ES is one of the basic data structures in programming languages semantics and implementation. It supports the *abstraction principle*, which allows the programmer to consider the currently being written piece of code to be independent of the contexts of its possible use. The stack makes it possible to associate parameters and local variables to a particular procedure (function, method, etc) invocation. Thus, safe nested calls of procedures from other procedures are possible, which includes recursive calls.

ES consists of *sections* that are sets of *binders*. A binder relates a name with a run time entity. Formally, it is a pair (n, x), where n is an external name ($n \in N$) and x is a reference to an object ($x \in I$); such a pair is written as $n(x)$. We refer to n as the *binder name* and to x as the *binder value*. The concept of a binder can be generalized – x can be an atomic value, a compound structure, or a reference to a procedure/method.

The process that determines the meaning of a name is called *binding*. Binding follows the "search from the top" rule i.e. to bind a name n the mechanism is looking for the ES "visible" section that is the closest to the top of ES and contains a binder with the name n. If the binder is $n(x)$, then the result of the binding is x. To cover bulk data structures of the store model, SBA assumes that binding can be multi-valued, that is, if the relevant section contains several binders whose names are n: $n(x_1)$, $n(x_2)$, $n(x_3)$,..., then all of them contribute to the result of the binding. In such a case the binding of n returns the collection $\{x_1, x_2, x_3, ...\}$.

At the beginning of a session ES consists of a base section that contains binders to all database root objects. Other base sections can contain binders to computer environment variables, to local objects of the user session, to libraries, etc. During query evaluation the stack grows and shrinks according to query nesting. Assuming there are no side effects in queries (i.e., no calls of updating methods), the final ES state is exactly the same as the initial one. Fig.2 presents this idea for the evaluation of the query "*Employee* where <predicate containing name *p*>" for the object store from Fig.1. The presented intermediate state concerns the binding of name *p* occurring in the *where* clause of the query; the search order is presented by the arrows.



attr_id(i12) attr_sex(i13)
eNo(i14) name(i15) job(i16) sal(i17)
rating(i18) works_in(i19)
Company(i1) Department(i2)
Employee(i11) Employee(i20)
Company(i1) Department(i2)
Employee(i11) Employee(i20)
Initial ES state
ES state during binding p
Final ES state
time

Fig. 2. States of ES during query evaluation

2.3 Stack-Based Query Language (SBQL)

In this section we present the general idea of SBQL, which is described in detail in [SBMS95, SKL95]. The language is implemented in the Loqis system [SMA90, Subi91], in the prototype of an XML query language for the DOM model and in the prototype for the ICONS project.

We argue that queries addressing XML need not be expressed through XML syntax, as suggested in [CFMR01]. There is no point in perceiving queries as XML files. In our opinion, XML syntax makes queries illegible. Legibility of queries is much more important for users than any other feature related to query syntax.

SBQL Syntax. SBQL is based on an abstract syntax and the principle of *compositionality* – it syntactically separates query operators. The syntax of the language is as follows:

- A single name or a single literal is an (atomic) query e.g. *Employee*, *y*, "Smith", 2.
- If q is a query and σ is a unary operator (e.g. sum, sqrt), then $\sigma(q)$ is a query.
- If q_1 and q_2 are queries and θ is a binary operator (e.g. where, dot, join, =, and), then $q_1 \theta q_2$ is a query.

SBA is based on the assumption of operator orthogonality – we can freely combine operators unless it violates some type constraints. SBQL divides operators into two categories: algebraic and non-algebraic. Now, we shortly present these two kinds.

Algebraic Operators. The operator is algebraic if it does not modify ES. Algebraic operators include string comparisons, Boolean and numerical operators, aggregate functions, set, bag and sequence operators and comparisons, the Cartesian product, etc. The evaluation of an algebraic operator is very simple: let $q_1 \Delta q_2$ be a query that consists of two subqueries connected by a symbol Δ denoting a binary algebraic operator Δ. First, q_1 and q_2 are evaluated independently; then Δ is performed on two partial results (taking them in the proper order), returning the final result.

Non-algebraic Operators. Non-algebraic operators include *selection*, *navigation*, *dependent join*, *quantifiers*, etc. If the query $q_1 \theta q_2$ involves a non-algebraic operator θ, then q_2 is evaluated in the context determined by q_1; thus the order of evaluation of sub-queries q_1 and q_2 is significant. This is the reason for which these operators are called "non-algebraic": they do not follow the basic property of algebraic expressions evaluation, which requires independent evaluation of q_1 and q_2.

The query $q_1 \theta q_2$ is evaluated as follows. For each element r of the query value (q_value) returned by q_1, the subquery q_2 is evaluated. Before each such evaluation ES is augmented with a new section determined by r. After the evaluation, the stack returns to the previous state. A partial result of the evaluation is a combination of r and the q_value returned by q_2 for that r; the kind of combination depends on θ. Next, these partial results are merged into the final result.

New ES section(s) are constructed and returned by a special function *nested*. For an element r the function is defined as follows:

- If r is a single identifier of a complex object, e.g. i_{11}, then *nested* returns binders to attributes (sub-objects) of a given object.
- If r is an identifier of a link object, e.g., i_8, then *nested* returns binders to the object the link points to (e.g. for i_8 it is $\{Employee(i_{11})\}$).
- If r is a binder, then *nested* returns $\{r\}$ (the set consisting of the binder).
- For r being a structure $struct\{r_1, r_2, r_3, ...\}$ *nested* returns the sum of results returned by *nested* for $r_1, r_2, r_3, ...$
- For other r *nested* returns the empty set.

Query Results. We call values returned by SBQL queries *q_values* and we define them in the following, recursive way:

- Each atomic value (e.g. 3, "Smith", TRUE, etc.) is a q_value.
- Each reference to an object (of any kind, e.g. i_{11}, i_{12}, i_{16}, i_{18}, etc.) is a q_value.
- If v is a q_value and n is any name, then a binder n(v) is a q_value.
- If v_1, v_2, v_3, ... are q_values and *struct* is a structure constructor, then *struct*{ v_1, v_2, v_3, ...} is a q_value. In general, the order of elements in the structure is essential. This constraint can be relaxed if all v_i are binders. This construct generalizes a tuple known from relational systems.
- If v_1, v_2, v_3, ... are q_values and *bag*, *sequence*, ... are collection constructors, then *bag*{ v_1, v_2, v_3,,... }, *sequence* { v_1, v_2, v_3, ... }, ... are q_values.
- There is no other q_values.

Note that there is no problem to map final result of our query to XML, providing all its sub-results are binders. However, it makes no sense to map intermediate query results to the XML format, because our semantics requires returning references to objects that are lost after such a conversion.

ES is closely related to definition of semantics. SBA uses another, auxiliary stack QRES (Query RESult) for storing results of (sub)queries in the operational style of semantics. In other definitions (e.g. denotational) QRES is not necessary.

Example Queries in SBQL[3]:

Company.Employee
Employee **where** *job* = "designer"
(*Department* **where** *count*(*employs*) > 20).*boss.Employee.name*

Procedures. SBQL incorporates procedures, with or without parameters, returning an output or not. A procedure parameter can be any query. We adopt *call-by-value*, *call-by-reference* and other parameter passing methods. There are no limitations on computational complexity, what can be useful for view definitions when the mapping between stored and imaginary objects is complex and requires non-trivial algorithms. The results of functional procedures (functions) belong to the same semantic category as results of queries, therefore they can be invoked in queries. It also means that there are no restrictions on calling functions within the body of (other) functions, what enables among others recursive calls. In current implementation procedures (and other features of SBQL) are untyped, but another group in our team intends to introduce static and dynamic type checking.

Below, we present an example function *underpaid* returning identifiers of employees whose salary is less then average and professions are listed in parameter *JobPar* (*JobPar* is a *call-by-value* parameter). Function has an auxiliary local variable *a*:

```
function underpaid ( in JobPar ) {
   local a := avg ( Employee . sal );
   return Employee where job = JobPar and sal < a; }
```

[3] Note that our queries are more "diabetic" than SQL and query languages for XML (XQL, Xquery, etc.): there is no "select", "from" and other sugar. This essentially improves orthogonality, compositionality and readability of nested queries. The sugar can be of course freely added to SBQL, according to taste.

We can call the function in a query that returns names of departments where underpaid clerks work:

underpaid(“clerk “) . *works_in* .*Department* . *dName*

Practically, in all the classical approaches (relational, object-relational and object-oriented) a view is essentially a functional procedure that is stored in a database. View updates are performed through side effects of view definitions, usually by various kinds of references to stored data (TID-s, OID-s, etc.) returned by view invocations. The art of view updating is focused on forbidding updates that may violate user intention, c.f. view updatability criteria, such as no over-updating of a view. For instance, to avoid over-updating in Oracle a user is allowed to update a virtual relation being a join of two stored relations, but updates can concern only attributes coming from the relation being on the foreign key side of the join. We definitely abandon this approach, disallowing any updates through side effects of view definitions. Instead, we introduce explicit information on intents of view updates in the view definition.

3 Updatable Views for XML

We formulated the following basic assumptions for our view mechanism:

- View definitions are complex entities in a spirit of abstract data types (but essentially different from ADTs), which define the information content of virtual objects together with all the required view updating operations. The operations overload generic updating operations on virtual objects and perform updates on stored data. The language for defining a content of virtual objects and view updating operations is based on SBQL and has full computational power of a programming language.
- Any view updating operation is fully in hands of a view definer. We assume no updating through side effects, e.g. by references returned by a view invocation.
- Preservation of the programming languages’ principles, such as semantic relativism, orthogonality and no exceptional or special cases. These principles support universality, conceptual simplicity, simplicity of the use, easy implementation and optimisability of the view mechanism. They also much reduce the size of documentation and learning time. Semantic relativism is an essential novelty of our view definitions, not present in previous approaches to views. It means that view definitions can be nested with unlimited number of nesting levels, up to “atomic” view definitions which define atomic virtual data. If a virtual object has attributes, each of them must be defined as a sub-view. Independently of the view hierarchy level, each view definition has the same syntax, semantics and pragmatics.
- Full transparency of virtual objects. When a view is defined, a user will be unable to recognize any difference in querying and manipulating virtual and stored objects.
- Universality. The approach that we propose will be applicable not only to XML-oriented databases, but also to object-oriented and relational databases. The approach can be easily extended to any kind complex object/data models.
- Queries involving views will be optimizable using query modification technique. Within the technique we can use all the optimization methods that have been developed for the given model (rewriting, indices, etc.).

In comparison to classical views (cf. SQL) we assume that the name of a view definition is different from the name of virtual objects determined by the view. Therefore, we explicitly introduce the *managerial name* of a view (used for operations on view definition) and a name of virtual objects (used for operations on virtual objects). We assume a simple naming convention where a managerial name has always the suffix "*Def*", e.g. *RichEmpDef*.

Now, we present main concepts of our updatable views mechanism. More detailed description can be found in [KLPS02].

3.1 View Definition

All current approaches to views take it for granted that the mapping between stored and virtual data/objects is determined by a single query. We consider such an approach inappropriate if one wants to introduce operations of view updates, because it causes a loss of semantic information. For instance, if a view returns (non-unique) boss names and average salaries in departments, then association of particular values returned by the view with particular departments can be lost; therefore, it would be impossible to write a view updating procedure that will require references to departments as parameters. Additionally, in this way we can take full control on any access to a view, including retrieval. A single query approach is a disadvantage of the (Oracle, SQL Server) "instead of" trigger views, making a lot of view updates impossible.

Therefore, we propose a two-query paradigm to view updates. The first query preserves all the necessary information about the stored source objects and the second query (wrapped in a procedure) takes the result of the first query as input and delivers the final mapping between stored and virtual objects. The first query returns a collection containing elements called seeds. A seed is used by the second query for making up a virtual object; the number of virtual objects is the same as the number of seeds. A seed is also used as a parameter of the updating procedures defined by the view definer for determining view updates. Passing this parameter is implicit (it is internal to the proposed mechanism). An entire virtual object growing up from a seed contains data determined by the second query, sub-views and the defined updating operations. There are no limitations concerning the complexity of a seed – it can be the result of queries involving joins or other query operators.

The result of the first query is the basis of the following operations:

- *Dereferencing* (the second query) that returns the value of a virtual object. This value can be complex: it can be composed of references, atomic values and names. It must be (usually implicitly) applied in situations where an identifier must be changed to value, e.g. the context of such algebraic operators as +, <, sum, call-by-value parameters, etc.
- *Updating*. The operation performs assignment to a virtual object. It has an r-value as a parameter.
- *Deleting*. The operator performs deleting of a virtual object.
- *Inserting*. The operator performs inserting a new (virtual) object to the inside of a virtual object. It has a reference to a new object as a parameter.

The syntax of a view definition is illustrated by the following example:

```
create view RichEmpDef {
    virtual objects RichEmp { return (Employee where salary > 1000) as r }
                                            // generating the collection of seeds
    on_retrieve do { ....... };  // dereferencing
    on_update rvalue do { ....... };     // assignment
    on_delete do { ....... };            // deleting
    on_insert objectref do { ....... };  // inserting
    .... //further text of the definition
}
```

Keywords **on_retrieve, on_update, on_delete, on_insert** are identical for all definitions of views. Each of the clauses is treated as a procedure. Names of parameters such as *rvalue* and *objectref* can be chosen by the view definer.

When a view is defined it is considered as a regular store object. The database section of the ES contains both: the binder with the reference to view definition (with managerial view name) and the second binder with virtual objects name (to indicate the existance of virtual objects). Both these binders are necessary. The first one is required to use or change the view's definition and the second one allows querying and updating virtual objects.

3.2 Virtual Identifiers

An essential issue concerns how to pass the information concerning an updating of a virtual object to the proper updating procedure defined within the view. Indeed, if a view invocation returns the result identical to the result of a query, then while performing the operation, the system is unable to recognize that the update concerns the view rather than regular stored objects. If the system cannot recognize it, it is unable to call the procedure that overloads the operation. The problem has been solved by the concept of *virtual identifier*, which is a triple:

<Flag "I am virtual", View definition identifier, Seed>

Virtual identifiers are counterparts of object identifiers. They are constructed in such a way that they deliver all the necessary information on virtual objects to view updating procedures written by the view definer; the system knows that the update concerns a view (due to the flag *"I am virtual"*), which view (due to the *View definition identifier*) and which virtual object (due to the *Seed*).

Processing of Virtual Identifiers. In SBA each non-algebraic operator processing an identifier pushes on ES all the binders referring to the interior of the object having this identifier. If an identifier is a virtual one of the form <*Flag "I am virtual", view_def_id, seed*>, then ES is first augmented by the section containing *nested*(*seed*) and next, by another section containing binders to all subviews of the *view_def_id* view. In such a way we pass the *seed* parameter to all the (dereferencing, updating) procedures that are defined within the view and, at the same time, we make all subviews (i.e. virtual attributes or sub-attributes) available for querying.

In Fig. 3 we present a general schema showing the control flow between different agents participating in serving view updates. Roughly, the scenario consists of:

1. Evaluation of a query invoking a view, which returns virtual identifiers.
2. Passing the virtual identifiers to the updating statement (containing the query).
3. Processing of the updating operation by the query/program interpreter. It recognizes that the operation concerns a virtual identifier. Thus, it makes no action on stored data, but passes the control to the proper procedure from the view definition. The interpreter prepares parameters for this procedure on the basis of seed stored within the virtual identifier.
4. Execution of the procedure, with an effect on stored data.
5. The control is passed back to the user program

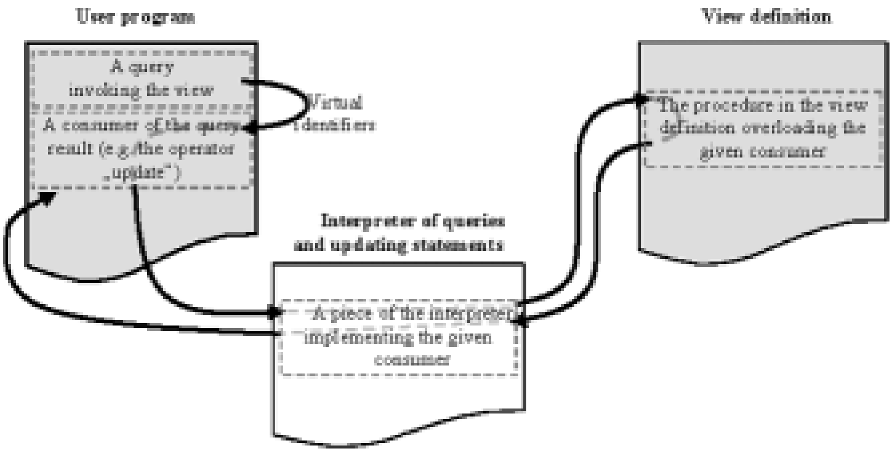

User program
A query invoking the view
A consumer of the query result (e.g. the operator „update")
Virtual identifiers
View definition
The procedure in the view definition overloading the given consumer
Interpreter of queries and updating statements
A piece of the interpreter implementing the given consumer

Fig. 3. Control flow during a view update

3.3 Nested Views

In our approach we can nest views, what implies that we have to extend a notion of virtual identifiers in order to access all seeds of its parent virtual objects along the path of nesting. A possible extended form of a virtual identifier is the following:

<Flag "I am virtual", (View definition identifier$_1$, Seed$_1$), ..., (View definition identifier$_n$, Seed$_n$) >

where (*View definition identifier$_1$, Seed$_1$*) refers to the most outer view, (*View definition identifier$_2$, Seed$_2$*) refers to its sub-view, etc., and (*View definition identifier$_n$, Seed$_n$*) refers to the currently processed view.

In a case when an identifier is processed by any of the procedures (*on_retrieve*, *on_update*, *on_delete*, or *on_insert*), the interpreter pushes on ES a section containing *nested*(*Seed$_1$*) ∪ *nested*(*Seed$_2$*) ∪ ... ∪ *nested*(*Seed$_n$*), and then calls the proper procedure. This is the way of passing information on seeds to all the procedures. The call requires identification of the proper procedure within nested views, e.g. according to the expression: *View definition identifier$_n$. on_update*

3.4 View Parameters and Recursive Views

In our approach views, similarly to procedures or functions, can have parameters. Parameters concern only a procedure generating seeds. In this paper we do not assume that a parameter will be available for definitions of updating procedures (it requires some minor extensions to the mechanism). A parameter is a query, with no restriction. In particular, it may return $bag(r_1, r_2, \ldots, r_n)$, where $r_1, r_2, \ldots, r_n$ are q_values. A query interpreter determines a method of parameters passing basing on view definition syntax. We are going to implement in our views the method of parameters passing known as *strict-call-by-value*, which combines *call-by-value* and *call-by-reference*. It means that the result of a query being a parameter is without any change passed to the function's body. Technically, it means that if for a formal parameter *par* the query returns the result *r* (however complex), then the corresponding activation record for function *f* is augmented by the single binder *par*(*r*). In this way the result of the query becomes available within the body of the function under name *par*.

Recursive views are side effects of SBA and its mechanisms. As shown, views are programming entities like functions or procedures. All volatile data created by the view are pushed on ES, thus each view call is independent on other view calls. Hence, recursive views are fully supported by the described mechanism.

3.5 Optimization

Due to full orthogonality and consistent conceptual continuation (with no anomalies and irregularities), queries involving views are fully optimizable through the query modification technique [Ston75], as presented in [SuPl01]. The optimization concerns cases when the procedure defining a collection of seeds is reduced to single (however complex) queries, with no parameters and recursion. These conditions are satisfied for majority of views. In all such cases textual substitution of the views invocation by the corresponding query from the procedure defining seeds results in a semantically equivalent query that can be optimized by standard methods, e.g. by removing dead sub-queries, factoring out independent sub-queries and low level techniques (e.g. based on indices). An example view optimization process can be found in [KLPS02].

4 Examples

In this section we present examples illustrating power of our view approach. Examples use source of XML data presented in Fig. 1.

Example 1: *Moving an employee to another department*

For each employee the view *EmpBoss*(*Surname*, *BossSurname*) returns a virtual object containing a pair of strings: the surname of an employee and the surname of their boss. The view should facilitate the following operations: (1) updating a boss name to *NewBoss* that causes moving the employee to the department managed by the *NewBoss*; (2) deleting of a virtual object that causes deleting the corresponding employee. Dereferencing procedures cause returning proper strings rather than references. No other operation is supported.

Note that we consciously update a view in the way, which according to updatability criteria of Oracle is forbidden (updating on the primary key side of the join). Because in our case such updating is reasonable, it is allowed.

```
create view EmpBossDef {
    virtual objects EmpBoss { return Employee as e }
    on_delete do { delete e }
    create view SurnameDef {
       virtual objects Surname { return e.name as es }
       on_retrieve do { return es ° "" }}     //concatenation with the empty string
    create view BossSurnameDef {              // enforces dereferencing
       virtual objects BossSurname {
          return e.worksIn.Department.boss.Employee.name as bs }
       on_retrieve do { return bs ° "" }
       on_update ( NewBoss ) do {
          e.works_in :=& Department where (boss.Employee.name) = NewBoss }}}
```

We assume automatic updating of twin pointers by the system, c.f. the ODMG standard: e.g. updating of *works_in* causes automatic updating of the twin *employs*.

Examples of view calls:

1. Get surnames of all employees working for Smith:

   (*EmpBoss* **where** *BossSurname* = "Smith") . *Surname*
2. Fire the employee named Cooper:

   delete *EmpBoss* **where** *Surname* = "Cooper";
3. Introduce a new boss Smith for employee Cooper:

   for each *EmpBoss* **where** *Surname* = "Cooper" **do** *BossSurname* := "Smith";
4. Change the surname of the employee named "Coper" to "Cooper":

   for each *EmpBoss* **where** *Surname* = "Coper" **do** *Surname* := "Cooper";

   ! Incorrect: *SurnameDef* does not support the updating operation.

Example 2: *Updating average salaries in departments*

A view *DeptAvgSal*(*DeptName, AvgSal*) returns a virtual object containing names and average salaries for all departments located in Warsaw. Updating the average salary causes distribution of the rise among all employees of the department proportionally to their previous salaries and to their assessment. Of course, there could be many other intentions of view update; each requires a specific *on_update* procedure.

```
create view DeptAvgSalDef {
  virtual objects DeptAvgSal{
   return (Department where loc="Warsaw") as d; }
  create view DeptNameDef {
    virtual objects DeptName { return d.dName as dn; }
    on_retrieve do { return dn ° ""; } }
  create view AvgSalDef {
    virtual objects AvgSal{ return avg( d.employs.Employee.sal ) as a; }
    on_retrieve do { return a; }
    on_update ( newAvgSal ) do {
      create local weightedSum := sum( d.employs.Employee.(sal*rating));
      create local ratio := (newAvgSal – a) * count( d.employs ) / weightedSum;
      for each d.employs.Employee do sal := sal + ratio*sal*rating; } } }
```

Using this view we can change average earnings in departments in Warsaw e.g. rise the average earnings in the Toys department by 100:

for each *DeptAvgSal* **where** *DeptName* = "Toys" **do** *AvgSal*:= *AvgSal*+ 100;

Note that the distribution of individual updates is proportional to earnings and ratings, but the final effect is that the average salary rise in the Toys department is 100. The example illustrates the power of our method: we are able to perform view updating operations considered earlier by many professionals as absolutely unfeasible. We show here that such updates are not only feasible, but could be reasonable and necessary for many applications, which require complex mappings of business ontologies.

5 Conclusion

We have presented a new approach to updatable views, which is based on the Stack-Based Approach. We introduced the view mechanism for XML, but the approach can be easily extended to more complex object oriented databases containing notion of classes, inheritance and dynamic roles. Currently, we are implementing the ideas on top of an already implemented query language SBQL for an XML native database. Afterwards, we are going to extend the implementation to cover more complex object store models, in particular some superset of the RDF data model. We have shown that our approach enables the users to define very powerful views, including views with parameters, with a local environment, recursive, with side effects, etc. The concept is consistent, relatively easy to implement, very simple to use and enabling optimization by powerful query modification methods.

XML views may have many practical applications in the Web context. Views with full computational power, as described in this paper, cover also various kinds of mediators, wrappers and adaptors. The idea of such powerful views is at least worth discussion in the database and Web communities.

References

[AAC+98] S. Abiteboul, B. Amann, S. Cluet, T. Milo, and V. Vianu. Active views for electronic commerce. Conf. sur les Bases Données, 1998.

[Abit00] S.Abiteboul. On Views and XML. Proc. of PODS Conf., 1999, 1–9

[AGM+97] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for Semistructured Data. Proceedings of the Workshop on Management of Semistructured Data, Tucson, Arizona, May 1997

[CFMR01] D. Chamberlin, P. Fankhauser, M. Marchiori, J. Robie. XML Query Requirements. W3C Working Draft 15, February 2001

[KL02] H. Kang, J. Lim: Deferred Incremental Refresh of XML Materialized Views. CAiSE 2002: 742–746

[KLPS02] H. Kozankiewicz, J. Leszczyłowski, J. Płodzie , and K. Subieta. Updateable Object Views. Institute of Computer Science of PAS, Report 950, 2002

[Lac01] Z. Lacroix. "Retrieving and Extracting Web data with Search Views and an XML Engine". Workshop on Data Integration over the Web, in conjunction with the 13th CAiSE Conf., Switzerland, 2001.

[LAW99] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating Structured and Semistructured Data. Proceedings of the 7th Intl. Conf. on Database Programming Languages, Kinloch Rannoch, Scotland, 1999

[SBMS95] K. Subieta, C. Beeri, F. Matthes, and J. W. Schmidt. A Stack Based Approach to Query Languages. Proc. of 2nd Intl. East-West Database Workshop, Klagenfurt, Austria, 1994, Springer Workshops in Computing, 1995.

[SKL95] K. Subieta, Y. Kambayashi, and J. Leszczyłowski. Procedures in Object-Oriented Query Languages. VLDB Conf., 182–193, 1995

[SMA90] K. Subieta, M. Missala, and K. Anacki "The LOQIS System. Description and Programmer Manual", Institute of Computer Science of PAS, Report 695, 1990

[Ston75] M. Stonebraker. Implementation of Integrity Constraints and Views by Query Modification. Proc. of SIGMOD Conf., 1975

[Subi91] K.Subieta. LOQIS: The Object-Oriented Database Programming System. Proc. 1st Intl. East/West Database Workshop on Next Generation Information System Technology, Springer LNCS 504, 1991, 403–421

[SuPl01] K. Subieta, J. Płodzie . Object Views and Query Modification. In Databases and Information Systems (eds. J. Barzdins, A. Caplinskas), Kluwer Academic Publishers, 2001, 3–14

[Wied92] G. Wiederhold. Mediators in the Architecture of Future Information Systems, IEEE Computer Magazine, March 1992

Testing Containment of XPath Expressions in Order to Reduce the Data Transfer to Mobile Clients

Stefan Böttcher and Rita Steinmetz

University of Paderborn
Faculty 5 (Computer Science, Electrical Engineering & Mathematics)
Fürstenallee 11, D-33102 Paderborn, Germany
{stb,rst}@uni-paderborn.de

Abstract. Within mobile client-server applications which access a server-side XML database, XPath expressions play a central role in querying for XML fragments. Whenever the mobile client can use a locally stored previous query result in order to answer a new query instead of accessing the server-side database, this can significantly reduce the data transfer from the server to the client. In order to check whether or not a previous query result can be reused for a new XPath query, we present a containment test of two XPath queries which combines two steps. At first, we use the DTD in order to check whether all paths selected by one XPath expression are also selected by the other XPath expression. Then we right-shuffle predicate filters of both expressions and start a subsumption test on them.

1 Introduction

1.1 Problem Origin and Motivation

Our work is motivated by the development of an XML database system which can be accessed by mobile clients. Whenever mobile clients with a small bandwidth connection to a server-side database query for XML data which is provided by the server, data transmission becomes a major cost factor. Whenever a client discovers that the data needed to answer the query is already stored in the client's memory, say as a previous query result, reusing the previous query result to answer a query will significantly save communication costs between server and client - in comparison to the submission of the new query to the server and the transfer of the results back to the client. It may be a significant advantage to reuse the local data in answering a query instead of resubmitting the data. Therefore, for each new XPath query, the client tries to find out whether or not the queried data is a subset of a previous query result – or as we say, the new XPath query *is subsumed by* a previous XPath query.

On the other hand, it is important that the client does not have to wait too long for the result of such a subsumption test, i.e., we have to find a balance between communication costs and the time accepted for a subsumption test. Therefore, we present an approach which can efficiently decide, whether or not one XPath query is subsumed by another XPath query for a given set of allowed XPath expressions. We allow however our tester to be incomplete, i.e., whenever our tester cannot efficiently decide whether or not one XPath query is subsumed by another one, we allow the sub sumption tester to return false, i.e., we assume that the previous query result cannot be reused. In this case, the query is sent to the database system. If however our algorithm returns true, we are then sure, that a previous query result can be reused. This paper focuses on the subsumption tester itself, whereas the reuse of old query results which are stored in the client's cache for a new query is discussed in [2], and concurrency control issues, e.g. how we treat a cached query result, when concurrent transactions modify the original XML database fragment, are discussed in [3].

1.2 Relation to Other Work and Our Focus

Our contribution is related to other contributions to the areas of semantic caching and containment tests for XPath. Within the field of semantic caching, we can distinguish at least the following major directions. There are contributions to server-side caching (e.g. [9]), caching in mediators, proxies or wrappers which build an intermediate layer between the client and the server (e.g. [4, 10, 11, 12]), caching in intranet servers with a fast connection to the clients (e.g. [15]), and the direction which we follow, i.e. contributions to client-side caching, e.g. ([5, 16, 17]).

The concept of semantic caching in contrast to tuple-based and page-based caching was introduced in [16]. While some contributions (e.g. [5]) focus on the architecture and replacement-strategies of a client-side cache, other contributions (e.g. [17]) add formal definitions of semantic caching and examine how new queries can be efficiently processed against the cache or examine the query containment on semi-structured data for other query languages (e.g. [8]).

In contrast to all the above contributions, we examine the XPath expressions themselves in order to decide whether or not the set of data selected by a new XPath query expression is a subset of the data selected by a previous XPath expression. For this purpose, we follow [6, 13, 14, 18], which also contribute to the solving of the containment problem for two XPath expressions under a given DTD.

The focus of the contributions [6, 13, 14, 18] is to find a general solution to the containment tests for certain subclasses of XPath expressions, and to report on decidability results or to give upper and lower bounds for the complexity of the containment test for certain subclasses of XPath expressions. However, in contrast to this, we focus on a fast decision as to whether or not an XPath query result can be reused and we allow our containment test to be incomplete.

The other contributions (e.g. [6, 13, 14, 18]) use tree patterns in order to normalize the XPath query expressions and to compare them to the structure of the database. They consider the DTD either as a set of constraints that has to be met [6, 18] or as an automaton [14]. In contrast to this, we follow [1] and use the concept of a DTD graph to expand all paths that are selected by an XPath expression, and we right-shuffle all filter expressions within sets of selected paths. Our transformation of an XPath query into a graph is similar to the transformation of an XPath query into an automaton, which was used in [7] to decide whether an XML document fulfills this query. In extension of our previous work [1], we allow for cycles in the DTD, we extend our approach to constraints given by the DTD, we generalize the concept of containment tests, and we use a new method of computing the containment of one path set within another. Finally, in contrast to all the other contributions, our approach combines the computation of path sets with the right-shuffling of predicate filters in such a way, that the containment test for XPath expressions can be reduced to a containment test of filters combined with a containment test of possible filter positions.

The remainder of the paper is organized as follows. Section 2 defines the problem, Section 3 outlines our approach to the computation of paths sets, Section 4 describes how our solution is extended to predicate filters, and Section 5 contains the summary.

2 Underlying Concepts and Problem Description

2.1 XPath Expressions for the Reuse of Previous Query Results

We assume that every access to XML data is described in terms of XPath expressions on the application side. For example, a query for customer information could be described by the following XPath expression

// customer [// address / city]

which means that within the XML document we query for any 'customer' node which has a descendant node 'address' which itself has a child node 'city'.

In general, an XPath expression is defined as being a sequence of location steps

/<LocationStep1> / … / <LocationStepN>,

where <LocationStepI> is defined as

axis-specifierI :: node-testI [predicate filterI],

however, wherever possible, we use the short-cut notation of the previous example.

2.2 The Subset of Considered XPath Expressions and DTD Declarations

Because XPath is a very rich language and we have to have a balance between the allowed complexity of XPath expressions and the complexity of the subsumption test algorithm on two of these XPath expressions, we restrict the set of XPath expressions to the following set of *allowed XPath expression*s:

1 Axis Specifiers

We allow *absolute* or *relative location paths* with the following *axis specifiers* in their *location steps*: self, child, descendant, descendant-or-self and attribute, and we forbid the parent, ancestor, ancestor-or-self, namespace, following (-sibling) and preceding (-sibling) axes.

2 Node Tests

We allow all *node name tests* except wildcards * and name-spaces, but we forbid *node type tests* like text(), comment(), processing-instruction() and node().

For example, when A is an attribute and E is an element, then ./@A, ./E, ./E/E, .//E are allowed XPath expressions.

3 Predicate Filters

We restrict *predicate filters* of allowed XPath expressions to be either a *simple predicate filter* [B] with a *simple filter expressions* B or to be a compound predicate filter.

3.1 Simple Filter Expressions

Let <path> be a relative XPath expression which uses the parent-axis, the attribute-axis or a child-axis location step, and let <value> be a constant.

- <path> is a simple filter expression. For example, when A is an attribute and E is an element, then @A and ./E are allowed filter expressions which are used to check for the existence of an attribute A or an element ./E respectively.
- Each *comparison* <path> = <value> and each *comparison* <path> != <value> is a simple filter expression.[1]
- 'not <path>' is a simple predicate filter, which means that the element or attribute described by <path> does not exist.

3.2 Compound Predicate Filters

If [B1] and [B2] are allowed predicate filters, then '[B1] [B2]', '[B1 and B2]', '[B1 or B2]' and '[not (B1)]' are also allowed predicate filters. In our subset of allowed predicate filters, '[B1] [B2]' and '[B1 and B2]' are equivalent, because we excluded the sibling-axes, i.e., we do not consider the order of sibling nodes.

2.3 Basic Definitions and Problem Description

We use an XPath expression XP = // E1 / E2 // E3 // E4 / E5 in order to explain some basic terms that we use throughout the rest of the paper. The *nodes selected by* XP (in this case elements with the node name E5) can be reached from the root node by a *path* (that must contain at least the nodes 'root', E1, E2, E3, E4, and E5). All these paths will be called *paths selected by XP* or *selected paths* for short.

Since every path to elements selected by XP contains an element E1 which is directly followed by an element E2, we call E1/E2 an *element sequence*. A single element (like E3) which does not require another element to directly follow it, will also be called an element sequence (consisting only of this single element). So, the element sequences in this example are: E1/E2, E3, and E4/E5.

The input of our tester consists of three parts: a DTD and two XPath expressions (XP1 and XP2) which are used for a query and a previous query result respectively. The goal is to prove, based on the DTD only, that the first XPath expression selects a subset of the node set selected by the second XPath expression. This should be done as efficiently as possible and without access to the actual XML database, because the subset test is completely executed on the client-side.

Throughout this paper, we use the term *XP1 is subsumed by XP2* as a short notation for "XP1 selects a subset of the node set selected by XP2 in all XML documents which are valid according to the given DTD". Given this definition, we can also state the subsumption test as follows: XP1 is subsumed by XP2, if and only if a path to an element which is selected by XP1 and is not selected by XP2 can *never* exist in any valid XML document.

[1] Note that we can also allow for more comparisons, e.g. comparisons which contain the operators '<', '>', '≤' or '≥' and comparisons like <path> <comparison operator> <path2>. In such a case the predicate filter tester described in Section 4.3. would have to check more complex formulas.

3 Overview and Key Concepts

Our goal is to check for two given XPath expressions XP1 and XP2, whether or not XP1 is subsumed by XP2 without any access to the data of the XML document.

We explain step by step our key ideas on how to check whether or not one XPath expression is subsumed by another. Firstly, we just consider XPath expressions without predicate filters. We delay the treatment of filters until Section 4.

3.1 The Basic Idea: Placing All Nodes of XP2 onto Each Path of XP1

In order to explain the basic idea, we consider XPath expressions XP1 and XP2 of the following form: XP1=//S11//S12//…//S1n, XP2=//S21//S22//…//S2m, where Sij are sequences of the form Sij=E11/…/E1k, and where E11, …, E1k are elements defined in the DTD of the XML document. In this case, XP1 is subsumed by XP2, if and only if each path selected by XP1 contains the sequences S21,…,S2m of XP2 in the correct order. In other words, the sequences of XP2 have to be placed somewhere in each path that is selected by XP1. Our basic idea is to use the given DTD in order to compute the set of all possible paths for XP1 in any valid XML document. Then, XP1 is subsumed by XP2, if and only if all paths for XP1 which are allowed by the DTD contain all sequences of XP2 in the correct order. In other words, if one path selected by XP1 which does not contain all sequences of XP2 in the correct order is found, then XP1 is not subsumed by XP2.

3.2 Constructing a DTD Graph

In order to find all paths selected by XP1 in the set of XML documents which are valid according to a given DTD, firstly we transform the DTD into a so called DTD graph and we then use this DTD graph to compute a so called XP1 graph which describes all paths selected by XP1. Within this paper, we use the concept of a directed DTD graph which was introduced in [1].

A *directed DTD graph* is a directed graph G=(N,C) where each node E∈N corresponds to an element of the DTD and an edge c ∈ C, c=(E1,E2) from E1 to E2 exists for each element E2 that is used to define the element E1 in the DTD. For example (**Example 1**), Figure 1 shows a DTD and the corresponding DTD-graph:

```
<!ELEMENT Root  (E1, E2)>
<!ELEMENT E1    (E2, E3)>
<!ELEMENT E2    (E1?)>
<!ELEMENT E3    (#PCDATA)>
```

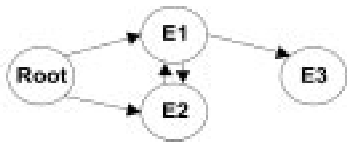


Fig. 1. DTD and corresponding DTD graph of Example 1

We use this DTD and two XPath expressions XP1 = //E3 and XP2 = //E1/E3 in order to illustrate the subsumption test. The first step (which is outlined in Section 3.5) is to find the set of all paths selected by XP1. In this example, the set of paths selected by XP1 can be written as

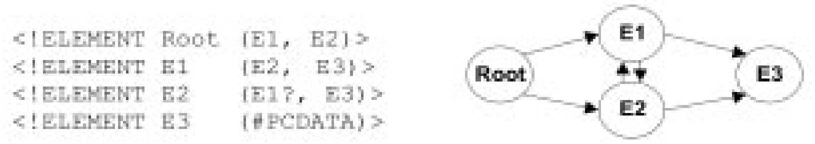
/Root(/E1/E2)^n/E1/E3 with $0 \le n < \infty$ | /Root(/E2/E1)^m/E3 with $1 \le m < \infty$

where '/Root(/E1/E2)^n/E1/E3 with $0 \le n < \infty$ ' is an abbreviation for the set of paths which contains the paths

/Root/E1/E3, Root/E1/E2/E1/E3, Root/E1/E2/E1/E2/E1/E3, …, etc. .

Similarly, ' (/E2/E1)^m with $1 \le m < \infty$ ' means that /E2/E1 has to occur at least once in each selected path, and | describes the union of two path sets. As we can see, each path selected by XP1 terminates with the sequence E1/E3, and therefore XP1 is subsumed by XP2.

However, if we change the DTD slightly to the following **Example 2**



```
<!ELEMENT Root  (E1, E2)>
<!ELEMENT E1    (E2, E3)>
<!ELEMENT E2    (E1?, E3)>
<!ELEMENT E3    (#PCDATA)>
```

Fig. 2. DTD and corresponding DTD graph of Example 2

we get an additional edge in the DTD graph and the set of selected paths changes to

/Root(/E1/E2)^n/E1/E3 with $0 \le n < \infty$ | /Root(/E2/E1)^m/E3 with $1 \le m < \infty$
| /Root(/E1/E2)^i/E3 with $1 \le i < \infty$ | /Root(/E2/E1)^j/E2/E3 with $0 \le j < \infty$.

This set of selected paths also contains the path /Root/E2/E3. However, this path does not contain the sequence E1/E3 which is required by XP2, and thus XP1 is not subsumed by XP2.

The DTD graph can be used to check whether or not at least one path selected by XP1 (or XP2 respectively) exists. If there does not exist any path selected by XP1 (or XP2 respectively), then XP1 (or XP2) selects the empty node set. We consider this to be a special case, i.e., if XP2 selects the empty node set, we try to prove that XP1 also selects the empty node set. For the purpose of the discussion in the following sections, we assume that at least one path for XP1 (and XP2 respectively) exists.

3.3 DTD Filters Represent Constraints Given by the DTD

The concept of the DTD graph represents an upper bound for the computation of possible paths for XP1, but it does not yet contain all the concepts of a DTD. In order to distinguish between optional child elements and mandatory elements and in order to distinguish between disjunctions and conjunctions found in the DTD, etc., we can associate additional DTD filters to each element of the DTD (and to each node of the DTD graph respectively). For example, a DTD rule

```
< !element E1 ( E2 ,( E3+ | E4+ ), E5? ) >
```

is translated into the following DTD filter for the element (or DTD graph node) E1:

[./E2 and (./E3 xor ./E4) and unique(E2) and unique(E5)] .

The relative path './E2' which occurs in the filter requires every element E1 to have a child node E2, and unique(E2) states that there can not be more than one child node E2 per node E1. Since E5 is an optional child of E1, its existence is not required. A complete tester would have to add the DTD filter for a node to each XPath expression which selects (or passes) this node. These DTD filters can be used as predicate filters to both XPath expressions whenever the edge is passed. These filters can also be used to eliminate so called *forbidden paths*, i.e. paths which are contained in the DTD graph, but which are not allowed when also filters or DTD filters are regarded. For example, all paths which require the existence of both children, E3 and E4, (e.g. //E1[./E3]/E4) are forbidden paths which can be discarded.

However, an incomplete tester may ignore some or even all of these DTD constraints for the following reason. The number of valid documents can only be increased and can never be decreased by ignoring a DTD constraint. When XP1 is subsumed by XP2 within this relaxed DTD (represented by the DTD graph) which allows for even more paths, then XP1 is also subsumed by XP2 under the more restrictive DTD which was originally given. Therefore, a successful proof of the subsumption of XP1 by XP2 within the relaxed DTD is sufficient for the reuse of a previous query result.

3.4 Element Distance Table in the DTD Graph

A further preparation step involves the use of the DTD graph to compute all possible distances for each pair of elements and to store these distances in a distance table, which we will call *DTD distance table* throughout the remainder of this paper. We use the distances for the right-shuffling of predicate filters in Section 4. For example, the distance from E1 to E2 in the DTD graph of Example 1 is any positive odd number of child-axis location steps, i.e., the distance table contains an entry "`2*x+1 (x≥0)`" which describes the set of all possible distances from E1 to E2. The distance table entry "`2*x+1 (x≥0)`" represents a *loop* of 2 elements (E1 and E2) in the DTD graph with a minimum distance of 1. Whenever two elements (E1,E2) occur in the DTD graph in a single loop which contains 'c' elements and the shortest path from E1 to E2 has the length k, then the DTD distance entry for the distance from E1 to E2 is 'c*x+k (x≥0)'. Since alternative paths or multiple loops which connect one element to another may exist, the general form of an entry of the DTD distance table is

$$\sum_{1 \le i \le n} ai*xi + k\ (xi \ge 0) \text{ or } \ldots \text{ or } \sum_{1 \le j \le m} bj*yj + p\ (yj \ge 0),$$

where ai,bj,k,p are natural numbers or 0.

Each name of a variable – x in the case of the previous example – is connected uniquely to one circle in the DTD graph and is called the *circle variable* of this circle.

3.5 Rolling Out the XP1 Graph (Algorithm 1)

The first part of our approach is the computation of the set of paths selected by XP1. We represent and store the set of selected paths in a graph, which we will call the *(rolled out) XP1 graph* in the remainder of the paper. Each path from the root node to the selected node in the rolled out XP1 graph describes one path selected by XP1, i.e. the set of paths selected by XP1 is equivalent to the set of paths from the root node to the selected node in the XP1 graph (as long as we ignore forbidden paths, i.e. paths that have predicate filters which are incompatible with DTD constraints and/or the selected path itself)[2]. We use the (rolled out) XP1 graph in order to check whether or not all of the paths selected by XP1 contain the sequences of XP2.

Example 3: Consider the DTD graph of Example 1 and an XPath expression 'XP1new=/root/E2/E1/E2//E3', which requires that all XP1 paths start with the element sequence /root/E2/E1/E2 and end with the element E3. The rolled out XP1 graph for the XPath expression XP1new is

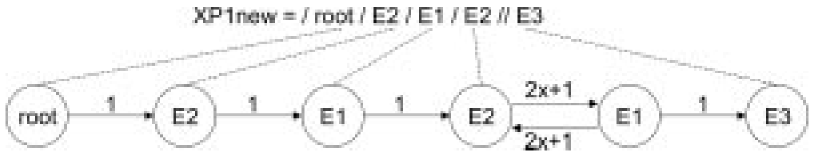


Fig. 3. XP1 graph of Example 3

where node labels visualize the nodes of the paths selected by XP1 and the edge labels visualize the possible distances between two nodes.

The (rolled out) XP1 graph depends on the DTD graph and on the given XPath expression, i.e., the XP1 graph for the DTD graph given in Section 3.2 and the XPath expression XP1 = //E3 is identical to the DTD graph given. The following algorithm computes the XP1 graph from a DTD graph given and an XPath expression XP1:

```
GRAPH GETXP1GRAPH(GRAPH DTD, XPATH XP1)
(1)   {
(2)     GRAPH XP1Graph = NEW GRAPH ( DTD.GETROOT() );
(3)     NODE lastGoal = DTD.GETROOT();
(4)     while(not XP1.ISEMPTY()) {
(5)       NODE goalElement = XP1.REMOVEFIRSTELEMENT();
(6)       if (XP1.LOCATIONSTEPBEFORE(goalElement) == '/')
(7)         XP1Graph.APPEND( NODE(goalElement) );
(8)       else
(9)         XP1Graph.EXTEND(
(10)          DTD.COMPUTEREDUCEDDTD(lastGoal,goalElement) );
(11)      lastGoal = goalElement;
(12)    }
(13)    return XP1Graph;
(14)  }
```

The algorithm generates a sequence of XP1 graph nodes for each element sequence of XP1. Furthermore, for each descendent-axis step E1//E2 which occurs in XP1, the algorithm inserts a subgraph of the DTD graph which represents all paths from E1 to E2 (which are allowed according to the DTD graph) between the nodes for E1 and E2. The method `COMPUTEREDUCEDDTD(lastGoal,goalElement)` returns such a subgraph of the DTD graph which only contains edges which are part of a path from `lastGoal` to `goalElement`. The result of a call of this function can

[2] With arbitrary filters and DTD constraints, the XP1 graph contains a superset of all paths selected by XP1, because some paths contained in the XP1 graph may be forbidden, e.g. because their filters contradict to DTD constraints.

be computed once in advance, i.e., before the evaluation of XP1, for each possible pair of nodes of the DTD graph, and it can be stored in a so called *table of reduced DTD graphs*, so that these subgraphs do not have to be computed during runtime.

As we must know the distances from each node to each other node in the XP1 graph during the treatment of the predicate filters, we compute the distances between adjacent nodes while processing the Algorithm 1 and we attach them as labels to the edges between them. There are two different types of edges: Edges that emerge through the method-call `XP1Graph.APPEND(NODE(goalElement))` which append only a single node. These edges always have the label "1". The other type of edges are edges inside the reduced DTD graphs. The distance between the first and the second node of these edges can be looked up in the DTD distance table.

3.6 Placing All XP2 Sequences Using the XP1 Graph (Algorithm 2)

The following algorithm which is used on an XPath expression XP2 and the XP1 graph tests whether or not all paths in the XP1 graph contain all sequences of XP2 (in the correct order). If we find at least one path from the root node to the selected node in the XP1 graph which does not contain all the element sequences of XP2 in the correct order (and this path is not a forbidden path), then XP1 is not subsumed by XP2.

Because we want to place XP2 element sequences in paths selected by XP1, we define the correspondence of XP2 elements and XP1 graph nodes as follows. An XP1 graph node and a node name which occurs in an XP2 location step *correspond* to each other, if and only if the node has a label which is equal to the element name of the location step.

We say, *a path (or a node sequence) in the XP1 graph and an element sequence of XP2 correspond* to each other, if the n-th node corresponds to the n-th element for all nodes in the XP1 graph node sequence and for all elements in the XP2 element sequence.

Before we start the algorithm, we transform XP2 into an equivalent normalized form by the following two steps. Firstly, relative XPath expressions are transformed into equivalent absolute XPath expressions. Secondly, if XP2 does not start with a child-axis location step, we insert '/root' at the beginning of XP2, i.e., in front of the first location step of XP2. 'root' corresponds to the root-node of the DTD graph (and the XP1 graph respectively), so that after this normalization all XPath expressions XP2 start with a child-axis location step '/root'.

The algorithm stated below firstly (line (3)) deals with the special case that XP2 consists of only one sequence, i.e. contains no descendant-axis location step. In this case, XP1 is subsumed by XP2, if the XP1 graph consists of only one path and this path corresponds to the sequence.

The main part of Algorithm 2 (starting at line (4)) solves the case that XP2 consists of more than one element sequence. A special treatment of the first sequence of XP2 (lines (6)-(8)) is required for the following reason. Because the first sequence of XP2 has to be placed in such a way that it starts at the root node and (if XP1 is subsumed by XP2) all paths selected by XP1 must start with this sequence, a prefix of each path which starts at the root node of XP1 graph must correspond to the first element sequence of XP2. This test is performed (at line (7)) by a call of the procedure
`BOOLEAN PLACEFIRSTSEQUENCE(in XP1Graph,inout XP2,inout startNode)`

which checks whether or not each path which begins at `startNode` has a prefix which corresponds to the first sequence of XP2, i.e., the node sequences that correspond to the first element sequence of XP2 can not be circumvented by any XP1 path. As more than one path may exist and therefore there may be more than one node which corresponds to the final node of this sequence, while testing this, the procedure internally stores the final node that is nearest to the end node of the XP1 graph (we call it the *last final node*).[3] If all paths which begin at `startNode` have a prefix which corresponds to the first sequence of XP2, the procedure `PlaceFirstSequence(…,…,…)` can successfully place the first XP2 sequence, therefore, it removes the first sequence from XP2, copies the last final node to the inout parameter `startNode` and returns `true`. Otherwise the procedure does `not` change XP2, does `not` change `startNode`, and returns `false`.

The middle sequences are placed by the middle part of the algorithm (lines (9)-(14)). In order to find the next possible `startNode` in the XP1 graph which corresponds to the first element of the currently first sequence of XP2, our algorithm calls a function `Search(in XP1Graph,in XP2,in startNode)`. As the next XP2 sequence must be contained in *each* path of the XP1 graph, the new `startNode` has to be only searched among the nodes which are common to all paths in the XP1 graph, i.e., the function `Search(…,…,…)` can skip the nodes that do not occur on each path from the previous `startNode` to the selected node. If such a next `startNode` does not exist, the function `Search(…,…,…)` then returns `null`.

Whenever the call of `Search(…,…,…)` returns a `startNode` (which is different to `null`), this `startNode` is then the candidate for placing the next XP2 sequence. That is why we call the procedure `PlaceFirstSequence(…,…,…)` again (line (12)). If this procedure call returns `false`, the current `startNode` is not a successful candidate for placing the sequence, and a call of `NextNode(…,…,…)` looks for the next node that is common to all paths in XP1 graph, as this is the first possible position of the new place for the XP2 sequence.

This middle part of the algorithm is continued until only one sequence remains in XP2. Similar to the first sequence of XP2, the last XP2 sequence is again a special case. However, this time it has to be ensured that each path which starts at the current `startNode` must have a *suffix* which corresponds to the last XP2 element sequence (if XP1 is subsumed by XP2), because the last XP2 sequence has to be placed in such a way that it terminates at the selected element of XP1 (line (15)).

This leads us to the following algorithm which decides in polynomial time whether or not each path selected by XP1 contains all sequences of XP2 in the correct order.

```
(1)   Boolean PlaceXP2(Graph XP1Graph, XPath XP2)
(2)   { if(XP2.ContainsOnlyOneSequence())
(3)        return (XP1Graph consists only of one path and
                   this path corresponds to XP2)
(4)     else        // XP2 contains multiple sequences
(5)     { //place the first sequence of XP2:
```

[3] When we place the next XP2 sequence at or 'behind' this last final node, we are then sure, that this current XP2 sequence has been completely placed before the next XP2 sequence, whatever path XP1 will choose.

```
       startNode:= XP1Graph.getROOT() ;
       if(not PLACEFIRSTSEQUENCE(XP1Graph,XP2,startNode))
          return false;
       // place middle sequences of XP2:
       while (XP2.containsMoreThanOneSequence())
       {  startNode:=SEARCH(XP1Graph,XP2,startNode);
          if(startNode == null) return false;
          if(not PLACEFIRSTSEQUENCE
                     (XP1Graph,XP2,startNode))
            startNode:=
                     NEXTNODE(XP1Graph,XP2,startNode);
       }
      //place last sequence of XP2:
       return (each path from startNode to the endNode
                has a suffix corresponding to XP2)
    }
  }
```

4 Including Predicate Filters (Algorithm 3)

Within this section we show how the above given algorithms must be extended so that they can also consider XPath expressions that contain predicate filters.

4.1 Normalizing XP1 Predicate Filters by Shuffling Them to the Right

In order to compare the predicate filters of the XPath expressions XP1 and XP2, we normalize them by shuffling their predicate filters to the right. The shuffling of all predicate filters to the right means that we modify each predicate filter so that it is attached to the last element of the XPath expression. Consider e.g. the DTD graph of Example 1 and an XPath expression XP1=/Root//E1[../@a="5"]/E3. Right-shuffling the predicate filter of XP1 yields the equivalent XPath expression XP1normalized= /Root//E1/E3[$(../)^2$@a="5"], where '$(../)^2$' is a short-cut notation for 2 parent-axis location steps and the exponent 2 is called the *distance of the simplified filter* [@a="5"] to the selected node, i.e., the simplified filter [@a="5"] must be applied to the grandparent of the selected element. The distance of a simplified XP1 filter to the selected element can simply be computed from the XP1 graph, i.e., it is equal to the distance from the node to which the filter is attached (E1 in the example) to the selected element (E3 in the example, i.e., the distance is 1) plus the number of parent axis-steps found in the filter expression (1 parent-axis step in [../@a="5"]).

Because the XP1 graph may contain loops, we describe node distances by the use of distance formulas which may contain variables and constants. Therefore, right-shuffling predicate filters usually yields distances formulas. For example, consider the following XPath expression

XP1 = /Root/E2/E1[@a="4"]/E2//E3

Since the distance of the 'left' node E1 to the selected node (E3) in the rolled out XP1 graph of Example 3 is described by the formula '2*x+3 (x≥0)', right-shuffling yields a normalized XPath expression

XP1normalized = /Root/E2/E1/E2//E3[(../)$^{2*x+3}$@a="4"] (x≥0).

4.2 Normalizing XP2 Predicate Filters

XP2 filters are right-shuffled in a similar fashion to XP1 filters. However, in order to compute the distance of a right-shuffled XP2 filter, we have to find a corresponding XP1 graph node to each element to which a filter is attached in XP2 and to compute the distance formula for the right-shuffled XP2 filter. For example (**Example 4**), if

XP2 = // E2 [../@a] // E3

then there are two nodes with label E2 in the XP1 graph of Example 3. If we match the location step //E2[../@a] of XP2 with the first (i.e. the left) E2 node in the XP1 graph, then the distance from the filter [../@a] to the selected node is 2*x+4 (x≥0). However, the second (i.e. the right) E2 node in the XP1 graph is part of a loop, i.e., when we match the location step //E2[../@a] with one of the E2 nodes represented by this loop, then the distance from the filter [../@a] to the selected node is 2*x'+2 (x≥x'≥0). Within this formula x expresses how many times XP1 can pass the loop (and thereby XP2 has to pass the loop in order to select the same node), and (x≥x'≥0) expresses that there are multiple occurrences of E2 to which the given location step of XP2 can assign its filter [../@a].
Altogether, we get the normalized XPath expression

XP2normalized = //E2//E3[(../)^{d}../@a] (d=2*x+4 and x≥0 or d=2*x'+2 and x≥x'≥0)

which can be simplified to

XP2normalized = //E2//E3[(../)^{d}@a] (d=2*x+5 and x≥0 or d=2*x'+3 and x≥x'≥0)

Because XP2normalized can chose x' to be equal to the value of x in the formula XP1normalized, i.e. XP2normalized can choose the same distance for its filter as the distance given in XP1normalized, and because the filter of XP1normalized is more specific than the filter of XP2normalized (i.e. @a="4" ⇒ @a), we now know that XP1 is subsumed by XP2.
In general, XP2 takes the form

/axis-specifier1::nodetest1[f1]/…/axis-specifierN::nodetestN[fN]

(the filter of a location step is omitted if the filter is equivalent to the [true]), and the equivalent normalized XPath expression takes the form

/axis-specifier1::nodetest1/…/axis-specifierN::nodetestN [(../)^{d1} f1]…[(../)^{dN} fN]

with d1, …, dN being the distance formulas.

In order to right-shuffle a filter from a given location step of XP2 to the selected node, we need to know the distances d1,…,dN from the location steps of XP2 which contain the filters to the selected node, or more precisely we need to know all the valid combinations of distances. For example, (**Example 5**) consider the XP1 graph given in Example 3 (as before) and an XPath expression XP2 = //E2[F2]//E1[F1]//E3. The node E1 in XP2 corresponds to two nodes E1 of the XP1 graph, i.e., the filter [F1] can be attached either to the first or to the second corresponding node E1 in the XP1 graph. Similarly, the filter [F2] which is associated with the element E2 in XP2 can be attached to either the first or the second corresponding node E2 in the XP1 graph. Note however, that XP2 must place the filter [F2] ‘before’ the filter [F1], i.e., it is not possible to attach the filter [F1] to the first occurrence of E1 in the XP1 graph *and* at the same time to attach the filter [F2] ‘behind’ this filter, i.e. to the second occurrence of E2 in the XP1 graph. That is why we further restrict the set of corresponding pairs of XP1 graph nodes and XP2 element names to the so called smaller *set of matching pairs of XP1 graph nodes and XP2 element names*.

Within a *set of matching pairs of XP1 graph nodes and XP2 element names* each XP2 element (and each XP2 location step respectively) is associated with a corresponding XP1 graph node, and all these corresponding pairs are in the correct order. That is, if the location step of an element E1 in XP2 precedes the location step of E2, then a *set of matching pairs* chooses the corresponding nodes E1 and E2 in the XP1 graph in such a way that a path from the chosen E1 node to the chosen E2 node in the XP1 graph exists. A set of matching pairs of XP1 graph nodes and XP2 elements can be computed from XP2 by simply restricting all pairs of corresponding XP2 elements and XP1 graph nodes to those only which are in the correct order.

When we compute the distance of a filter of an XP2 location step (like [F2] in Example 5) to the selected node (E3), we only have to consider paths via the node E1, because E1 is required by the next location step. Therefore, if we know the distance from the XP1 graph node, which corresponds to this E1 element, to the selected node, we can add the distance from E2 to E1 in order to get the distance from E2 via E1 to the selected node.

More general: Assume we have a set of matching pairs of XP1 graph nodes and XP2 elements (or location steps respectively) and we want to compute the distance formulas for the right-shuffling of the filters attached to the location steps. Then, in order to compute the distance from an arbitrary XP2 location step to the selected node, we must only consider XP1 graph paths via the nodes which correspond to the XP2 location steps that follow the arbitrary location step. This is, the distance can be summed up from location step to location step, i.e., the distance from an arbitrary location step to the selected node is equal to the sum of the distance from its matching node to the matching node of the next location step and the distance from the next location step to the selected node.

As Example 4 illustrates, in general there are many different sets of matching pairs of XP1 graph nodes and XP2 elements – in other words there are many possibilities where XP2 can place its filters in the XP1 graph. For a successful subsumption test it is sufficient to find at least one set of matching pairs of XP1 graph nodes and XP2 elements, where each filter of XP2 subsumes a filter of XP1.

Depending on the filter distance to the selected element and on the filter expression itself, it can be decided whether or not one filter is subsumed by another filter by using our filter tester which is described in the following section.

4.3 An Implication Test for Filter Expressions of Normalized Predicate Filters

The input of this third step are the normalized predicate filters for XP1 and XP2 which are computed by performing the steps described in the previous two sections.

If [f1] is the only filter of XP1 and [f2] is the only filter of XP2, then XP1 is subsumed by XP2, if and only if [f1] is at least as restrictive as [f2], i.e., f1⇒f2. For example, if the input contains only two filters [f1]=[../@a=”5”] and [f2]=[../@a], then

[f1] is more restrictive than [f2], because the implication f1⇒f2 holds, but f2⇒f1 does not hold.

Let d1 be a distance formula of a filter of XP1 and d2 be a distance formula of a filter in a location step of XP2. Both distance formulas d1 and d2 depend on (zero ore more) circle variables x1,…,xn of the XP1 graph, and d2 may additionally depend on an `x'` where (x≥x'≥0) and x is equal to one of the circle variables x1, …, xn. We say, *a loop* loop1=$(../)^{d1}$ *is subsumed by a loop*[4] loop2=$(../)^{d2}$, if all distances which are selected by loop1 are also selected by loop2. More specifically, loop1 is subsumed by loop2, if for all possible tuples of values for [x1, …, xn]. If d2 additionally depends on an x', loop1 is subsumed by loop2, if an x' can be found so that d1=d2 holds. Whenever a loop1 of a filter [f1i] of XP1 is subsumed by a loop2 of a filter [f2j] of XP2, then XP2 can place its filter [f2j] in such a way, that it is applied to the same elements as [f1i].

However, let XP1 be //E3 [$(../)^{2*x+1}$@a] (x≥0) and XP2 be //E3 [$(../)^{2*x+3}$@a] (x≥0), i.e., the loop 2*x+1 is *not* subsumed by the loop 2*x+3, then XP1 can choose x=0 (i.e. place its filter [@a] to the parent of E3), but XP2 has to place its filter [@a] to some previous ancestor element, say Ep. Because XP1 has no filter placed on Ep, XP1 includes also elements Ep which do not have an attribute 'a', i.e., XP1 is not subsumed by XP2.

For a successful subsumption test, at least one set of matching pairs of XP1 graph nodes and XP2 elements must exist such that the following holds: for each predicate filter [f2j] of XP2 there exists a predicate filter [f1i] of XP1 such that fexp1i⇒fexp2j, and the distance loop of f1i is subsumed by the distance loop of f2i (i.e. XP2 can place the filter [f2j] in such a way that it is applied to the same elements as the filter [f1i] of XP1). If this condition holds, the filter tester returns `true`. However, if no set of matching pairs of XP1 graph nodes and XP2 elements is found for which this condition holds, the tester returns `false`.[5]

Since both filter expressions that occur in the implication fexp1i⇒fexp2j contain no loops, we can use a predicate tester for XPath expressions (e.g. [1]), which extends a theorem prover for Boolean logic so that it takes the special features of simple XPath expressions into account (e.g. the tester has to consider that [not ./@a="5" and not ./@a!="5"] is equivalent to [not ./@a]).

4.4 An Extended Example

We complete this section with an extended example that includes all three major steps of Algorithm 3. Consider the DTD of Example 1 and the following XPath expressions
XP1 = / Root / E2[./@b] / E1[./@c=7] / E2 // E3[../../@a=5] and
XP2 = // E1[./../@b] // E2[./@a] // E3 (**Example 6**).

[4] The definition *one loop is subsumed by another* also includes paths without a loop, because distances can be a constant value.

[5] Here the algorithm is incomplete but one-sided correct, i.e. whenever the tester returns true, we know that XP1 is subsumed by XP2. Further details are omitted due to space limitations.

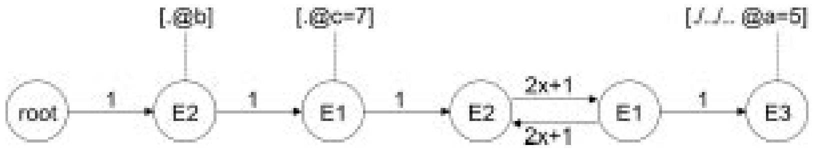


Fig. 4. XP1 graph of Example 6

Step1: The XP1 graph is computed. Figure 4 shows which filters of XP1 are attached to which XP1 graph nodes.

Step2: Algorithm 2 is started. As each path of the XP1 graph contains the nodes E1, E2, and E3, it returns `true`.

Step3: The filters are shuffled to the right. We get the following filters for XP1: $[../../@a=5][(../)^{2x+3}@c=7][(../)^{2x+4}@b]=[(../)^{2}@a=5][(../)^{2x+3}@c=7][(../)^{2x+4}@b],(x\geq 0)$.

We get two possible sets of matching pairs of the elements E1 and E2 of XP2 and therefore two different sets of distance formulas for the predicate filters of XP2.
When we match the element E1 with the 'left' node E1 of the XP1 graph, we get the following filter expression for XP2

$[(../)^{2x'+2}\ ./@a][(../)^{1+(2x+2)}./..@b] = [(../)^{2x'+2}\ @a][(../)^{2x+4}\ @b]\ (x\geq x'\geq 0)$.

The implication tester for predicate filters returns `true`, because for this first given match the following holds. The loop $(../)^{2x'+2}$ of the first filter of XP2 subsumes the loop $(../)^{2}$ of the first filter of XP1 and $@a=5\Rightarrow @a$ holds for the filter expressions of XP1 and XP2, furthermore, the loop $(../)^{2x+4}$ of the second filter of XP2 subsumes the loop $(../)^{2x+4}$ of the third filter of XP1 and $@b\Rightarrow @b$. Therefore the result of the complete test is `true`, i.e., XP1 is subsumed by XP2. XP1 therefore selects a subset of the data selected by XP2.

5 Summary and Conclusions

In order to reuse previous query results that are cached on mobile clients, we have developed a tester that checks whether or not a new XPath query XP1 is subsumed by the query XP2 of a cached previous query result. Different from other contributions to the XPath containment problem, we transform the DTD into a DTD graph, and we use this DTD graph in order to compute the so called XP1 graph, i.e., a graph which contains all paths selected by XP1. This allows us to almost completely separate the subsumption test on paths from the subsumption test on normalized filter expressions, which are computed by right-shuffling predicate filters to the selected node of the XP1 graph.
Our subsumption test on normalized predicate filters tries to find an equivalent or more specific filter of XP1 for each filter of XP2, whereas two predicate filters can be compared by independently examining the filter distance formulas and the filter expressions of these predicate filters. Finally, the choice of the implication tester for filter expressions is independent of our other ideas, i.e., we can choose a powerful but complex tester in order to test a larger subset of XPath expressions or we can choose a fast predicate tester, which is either incomplete or covers only a smaller subset of XPath.

For reasons of simplicity, we presented the algorithm in such a way that it strictly separates the subsumption test for path sets from the subsumption test for predicate filters. Nevertheless, it is possible to intermix both steps, e.g., to perform an implication test prior to the full expansion of an element path set – which yields further optimization possibilities. Furthermore, the results presented here seem to be not limited to DTDs, i.e., we consider the transfer of the presented results to XML Schema to be a challenging research topic.

References

[1] S. Böttcher, A. Türling: Access Control and Synchronization in XML Documents. In Proceedings XML Technologien für das Semantic Web – XSW, Berlin, 2002, LNI 14.

[2] S. Böttcher, A. Türling: XML Fragment Caching for Small Mobile Internet Devices. 2nd International Workshop on Web-Databases. Erfurt, Oktober, 2002. Springer, LNCS 2593, Heidelberg, 2003.

[3] S. Böttcher, A. Türling: Transaction Validation for XML Documents based on XPath. In: Mobile Databases and Information Systems. Workshop der GI-Jahrestagung, Dortmund, September 2002. Springer, Heidelberg, LNI-Proceedings P-19, 2002.

[4] Boris Chidlovskii, Uwe M. Borghoff: Semantic caching of Web queries, The VLDB Journal (2000) 9: 2–17

[5] Shaul Dar, Michael J. Franklin, Björn Þór Jónsson, Divesh Srivastava, Michael Tan: Semantic Data Caching and Replacement. VLDB 1996: 330–341

[6] Alin Deutsch, Val Tannen: Containment and Integrity Constraints for XPath. KRDB 2001

[7] Yanlei Diao, Michael J. Franklin: High-Performance XML Filtering: An Overview of YFilter, IEEE Data Engineering Bulletin, March 2003

[8] Daniela Florescu, Alon Y. Levy, Dan Suciu: Query Containment for Conjunctive Queries with Regular Expressions. PODS 1998: 139–148

[9] Vagelis Hristidis, Michalis Petropoulos: Semantic Caching of XML Databases. WebDB 2002

[10] Dongwon Lee, Wesley W. Chu: Semantic Caching via Query Matching for Web Sources. CIKM 1999: 77–85

[11] Luo Li, Birgitta König-Ries, Niki Pissinou, Kia Makki: Strategies for Semantic Caching. DEXA 2001: 284–298

[12] Qiong Luo, Jeffrey F. Naughton: Form-Based Proxy Caching for Database-Backed Web Sites. VLDB 2001: 191–200

[13] Gerome Miklau, Dan Suciu: Containment and Equivalence for an XPath Fragment. PODS 2002: 65–76

[14] Frank Neven, Thomas Schwentick: XPath Containment in the Presence of Disjunction, DTDs, and Variables. ICDT 2003: 315–329

[15] Chris Olston, Jennifer Widom: Best-effort cache synchronization with source cooperation. SIGMOD Conference 2002: 73–84

[16] Qun Ren, Margaret H. Dunham: Semantic Caching and Query Processing, SMU Technical Report 98-CSE-04

[17] Qun Ren: Semantic Caching in Mobile Computing. PhD thesis, Southern Methodist University, Computer Science and Engineering, Dallas, TX 75205, February 2000.

[18] Peter T. Wood: Containment for XPath Fragments under DTD Constraints. ICDT 2003: 300–314.

Query Containment with Negated IDB Predicates

Carles Farré, Ernest Teniente, and Toni Urpí

Universitat Politècnica de Catalunya
08034 Barcelona, Catalonia
{farre,teniente,urpi}@lsi.upc.es

Abstract. We present a method that checks Query Containment for queries with negated IDB predicates. Existing methods either deal only with restricted cases of negation or do not check actually containment but uniform containment, which is a sufficient but not necessary condition for containment. Additionally, our queries may also contain equality, inequality and order comparisons. The generality of our approach allows our method to deal straightforwardly with query containment under constraints. Our method is sound and complete both for success and for failure and we characterize the databases where these properties hold. We also state the class of queries that can be decided by our method.

1 Introduction

Query Containment (QC) is the problem concerned with checking whether the answers that a query obtains are a subset of the answers obtained by another query for every database. QC was first studied for the class of conjunctive queries [CM77]. QC of conjunctive queries with order comparisons was studied in [Klu88, Ull97]. Conjunctive QC with safe negated EDB atoms was investigated in [Ull97, WL03]. EDB stands for *extensional database*, that is, the database's stored relations whereas IDB means *intensional database*, that is, the relations constructed by deductive rules.

The methods that deal with negated IDB subgoals can be classified into two different approaches. The first one is to check QC for query classes where negation is used in a restrictive way [LS95]. The second approach is not to check "true" QC but another related property called *Uniform* QC [LS93], which is a sufficient but not necessary condition for QC [Sag88].

When considering integrity constraints, the containment relationship between two queries does not need to hold for any state of the database but only for those that satisfy the integrity constraints. This idea is captured by the notion of Query Containment under Constraints (QCuC). QCuC for datalog queries, without negation, and integrity constraints expressing tuple-generating dependencies was addressed in [Sag88] by taking the uniform containment approach. QCuC was also handled in the context of hybrid systems combining conjunctive queries and constraints expressed in a Description Logic language [LR96, CDL98].

In [FTU99] we sketched a method, named Constructive Query Containment method (CQC for short), to check "true" QC and QCuC in the presence of negation on IDB subgoals. Intuitively, the aim of our CQC method was to construct a *counterexample* that proves that there is no QC (or QCuC). This method used different *Variable Instantiation Patterns (VIPs)*, according to the syntactic properties of the queries and the databases considered in each test. Such a customization only affects the way that the facts to be part of the counterexample are instantiated. The aim was to prune the search of counterexamples by generating only the relevant facts.

We extend here our previous work by:

- providing not just an intuitive idea but also the full formalization of the CQC method.
- proving two additional theorems that hold when there are no recursively defined IDB relations: *failure soundness*, which guarantees that containment holds if the method terminates without building any counterexample; and *failure completeness,* which ensures that if containment holds between two queries then our method fails finitely (and terminates).
- ensuring termination when checking containment for conjunctive queries with safe EDB negation and built-in literals.
- pointing out that the CQC method is not less efficient than other methods that deal with conjunctive queries with or without safe EDB negation. We propose an additional VIP, the simple VIP, to perform such a comparison.
- decomposing the General VIP in two: the discrete order VIP and the dense order VIP that allow us to deal with built-in literals assuming both discrete and dense order domains.

It follows from these new results that the method we describe here improves previously proposed algorithms since it provides an efficient decision procedure for known decidable cases and can also be applied for more general forms of queries that were not handled by previous algorithms. In these more general cases, our method is semidecidable because it cannot be guaranteed termination under the presence of infinite counterexamples. Nevertheless, if there is a finite counterexample our method finds it and terminates and if containment holds our method fails finitely and terminates, provided that there are no recursively defined IDB relations in both cases.

Section 2 sets the base concepts used through the paper. In Section 3, we introduce our method and Section 4 formalizes it. In Section 5, we present the main correctness results of our method. For a more detailed formalization and detailed proofs, we refer to [FTU02]. In Section 6, we discuss the decidability issues regarding our method. In Section 7, we compare our method with related work. The paper ends with the conclusions, Section 8, and references.

2 Base Concepts

A *deductive rule* has the form:

$$p(\bar{X}) \leftarrow r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \neg r_{n+1}(\bar{Y}_1) \wedge \ldots \wedge \neg r_m(\bar{Y}_s) \wedge C_1 \wedge \ldots \wedge C_t$$

where p and $r_1, \ldots, r_m$ are *predicate* (also called *relation*) names. The atom $p(\bar{X})$ is called the *head* of the rule, and $r_1(\bar{X}_1), \ldots, r_n(\bar{X}_n), \neg r_{n+1}(\bar{Y}_1), \ldots, \neg r_m(\bar{Y}_s)$ are positive and negative *ordinary literals* in the body of the rule. The tuples $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n, \bar{Y}_1, \ldots, \bar{Y}_s$

contain *terms*, which are either variables or constants. Each C_i is a *built-in literal* in the form of $A_1\ \theta\, A_2$, where A_1 and A_2 are terms. Operator θ is $<$, $\leq$, $>$, $\geq$, $=$ or $\neq$. We require that every rule be *safe*, that is, every variable occurring in $\bar{X}$, $\bar{Y}_1$, …, $\bar{Y}_s$, C_1 , … or C_t must also appear in some $\bar{X}_i$.

The predicate names in a deductive rule range over the *extensional database* (EDB) predicates, which are the relations stored in the database, and the *intensional database* (IDB) predicates (like *p* above), which are the relations defined by the deductive rules. EDB predicates must not appear in the head of a deductive rule.

A set of deductive rules *P* is *hierarchical* if there is a partition $P = P_1 \cup \ldots \cup P_n$ such that for any ordinary atom $r(\bar{X})$ occurring positively or negatively (as $\neg r(\bar{X})$) in the body of a clause in P_i, the definition of *r* is contained within P_j with $j < i$. Note that a hierarchical set of deductive rules contains no recursive IDB relations.

A *condition* has the denial form of:

$$\leftarrow r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \neg r_{n+1}(\bar{Y}_1) \wedge \ldots \wedge \neg r_m(\bar{Y}_s) \wedge C_1 \wedge \ldots \wedge C_t$$

where $r_1(\bar{X}_1)$, …, $r_n(\bar{X}_n)$, $\neg r_{n+1}(\bar{Y}_1)$, …, $\neg r_m(\bar{Y}_s)$ are (positive and negative) ordinary literals; and C_1, …, C_t are built-in literals. We require also that every variable occurring in $\bar{Y}_1$, …, $\bar{Y}_s$, C_1 , … or C_t must also appear in some $\bar{X}_i$. Roughly, a condition in denial form expresses a prohibition: a conjunction of facts (literals in the body) that must no hold on the database all at once. Therefore, a condition is *violated* (*not satisfied*), whenever $\exists \bar{Z}\, (r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \neg r_{n+1}(\bar{Y}_1) \wedge \ldots \wedge \neg r_m(\bar{Y}_s) \wedge C_1 \wedge \ldots \wedge C_t)$ is true on the database, where $\bar{Z}$ contains the variables occurring in $\bar{X}_1$, …, $\bar{X}_s$, $\bar{Y}_1$, …, $\bar{Y}_s$, C_1 , … and C_t.

A *query Q* is a finite set of deductive rules that defines a dedicated n-ary *query predicate q*. Without loss of generality, other predicates than *q* appearing in *Q* are EDB or IDB predicates.

A query Q_1 is *contained* in a query Q_2, denoted by Q_1 S Q_2, if the set of answers of $Q_1(D)$ is a subset of those of $Q_2(D)$ for any database *D*. Moreover, Q_1 *is contained in* Q_2 *wrt IC*, denoted by Q_1 S$_{IC}$ Q_2, if the set of answers of $Q_1(D)$ is a subset of those of $Q_2(D)$ for any database *D* satisfying a finite set *IC* of conditions (*integrity constraints*).

3 The Constructive Query Containment (CQC) Method

The containment relationship between two queries must hold for the whole set of possible databases in the general case. A suitable way of checking QC is to check the lack of containment, that is, to find just one database where the containment relationship that we want to check does not hold: Q_1 *is not contained in* Q_2, written Q_1 c Q_2, if there is at least one database *D* such that $Q_1(D) \not\subset Q_2(D)$.

Given Q_1 and Q_2 two queries, the CQC method is addressed to construct the extensional part of a database (EDB) where the containment relationship does not hold. It requires two main inputs: the *goal* to attain and the s*et of conditions to enforce*. Initially, the goal is defined $G_0 = \leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)$, meaning that we want to construct a database where $(X_1, ..., X_n)$ could be instantiated in such a way that $q_1(X_1, ..., X_n)$ is true and $q_2(X_1, ..., X_n)$ is false. The set of conditions to enforce is $F_0 = \emptyset$, since there is no initial integrity constraint to take care about.

When considering a set *IC* of integrity constraints, we say that Q_1 *is not contained in* Q_2 *wrt IC*, written $Q_1 \mathsf{c}_{IC} Q_2$, if there is at least one database *D* satisfying *IC*, such that $Q_1(D) \not\subset Q_2(D)$. In this case, the EDB that the CQC method has to construct to refute the containment relationship must also satisfy the conditions in *IC*. This is guaranteed by making the initial set of conditions to enforce $F_0 = IC$ together with the goal $G_0 = \leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)$.

3.1 Example: $Q_1 \mathsf{c}\, Q_2$

The following example is adapted from the one in [FTU99] by introducing double negation on IDB predicates. It allows illustrating the main ideas of our method and to show its behavior under these complex cases. Let Q_1 and Q_2 be two queries:

$Q_1 = \{ sub_1(X) \leftarrow emp(X) \wedge \neg chief(X) \}$

$Q_2 = \{ sub_2(X) \leftarrow emp(X) \wedge \neg boss(X) \}$

where *emp* is an EDB predicate and *chief* and *boss* are IDB predicates defined by a set *DR* of deductive rules:

$DR = \{boss(X) \leftarrow worksFor(Z, X)$

$chief(X) \leftarrow worksFor(Y, X) \wedge \neg boss(Y) \}$

where *worksFor* is another EDB predicate.

Intuitively, we can see that Q_1 is less restrictive than Q_2 because Q_2 does not retrieve those employees having anyone working for them, while Q_1 allows retrieving employees having some boss working for them. Hence, we can find a database containing EDB relations such as *emp*(joan), *worksFor*(mary, joan) and *worksFor*(ann, mary), where sub_1(joan) is true but sub_2(joan) is false (*chief*(joan) is false because *boss*(mary) is true whereas *boss*(joan) is true). Therefore, Q_1 is not contained in Q_2. Note that an even smaller EDB containing just *emp*(joan) and *worksFor*(joan, joan) would have lead us to the same conclusion.

A CQC-derivation that constructs an EDB that proves $Q_1 \mathsf{c}\, Q_2$ are shown in Fig. 1. Each row on the figure corresponds to a CQC-node that contains the following information (columns):

1. The goal to attain: the literals that must be made true by the EDB under construction. When the goal is [] it means that no literal needs to be satisfied. Here, the initial CQC-node contains the goal $G_0 = \leftarrow sub_1(X) \wedge \neg sub_2(X)$. That is, we want the CQC method to construct a database where exists at least a constant k such that both sub_1(k) and $\neg sub_2$(k) are true
2. The conditions to be enforced: the set of conditions that the constructed EDB is required to satisfy. Recall that a condition is violated whenever all of its literals are evaluated as true. Here, the initial CQC-node contains the set of conditions to enforce $F_0 = \emptyset$.
3. The EDB under construction. Initial CQC-Nodes always have empty EDBs.
4. The conditions to be maintained: a set containing those conditions that are known to be satisfied in the current CQC-Node and that must remain satisfied

until the end of the CQC-derivation. Initial CQC-Nodes have always this set empty.

5. The account of constants introduced in the current and/or the ancestor CQC-nodes to instantiate the EDB facts in the EDB under construction. Initially, such a set contains always the constants appearing already in $DR \cup Q_1 \cup Q_2 \cup G_0 \cup F_0$.

	Goal to attain	*Conditions to enforce*	*EDB*	*Conditions to mantain*	*Used constants*
	$\leftarrow \underline{sub_1(X)} \wedge \neg sub_2(X)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1:A1	$\leftarrow \underline{emp(X)} \wedge \neg chief(X) \wedge \neg sub_2(X)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2:A2	$\leftarrow \underline{\neg chief(0)} \wedge \neg sub_2(0)$	$\emptyset$	$\{emp(0)\}$	$\emptyset$	$\{0\}$
3:A3	$\leftarrow \underline{\neg sub_2(0)}$	$\{\leftarrow chief(0)\}$	$\{emp(0)\}$	$\emptyset$	$\{0\}$
4:A3	[]	$\{\leftarrow \underline{chief(0)}, \leftarrow sub_2(0)\}$	$\{emp(0)\}$	$\emptyset$	$\{0\}$
5:B1	[]	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y), \leftarrow sub_2(0)\}$	$\{emp(0)\}$	$\emptyset$	$\{0\}$
6:B2	[]	$\{\leftarrow \underline{sub_2(0)}\}$	$\{emp(0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
7:B1	[]	$\{\leftarrow \underline{emp(0)} \wedge \neg boss(0)\}$	$\{emp(0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
8:B2	[]	$\{\leftarrow \underline{\neg boss(0)}\}$	$\{emp(0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
9:B3	$\leftarrow \underline{boss(0)}$	$\emptyset$	$\{emp(0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
10:A1	$\leftarrow \underline{worksFor(Z,0)}$	$\emptyset$	$\{emp(0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
11:A2	[]	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{emp(0), worksFor(0,0)\}$	$\emptyset$	$\{0\}$
12:B2	[]	$\{\leftarrow \underline{\neg boss(0)}\}$	$\{emp(0), worksFor(0,0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
13:B3	$\leftarrow \underline{boss(0)}$	$\emptyset$	$\{emp(0), worksFor(0,0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
14:A1	$\leftarrow \underline{worksFor(Z,0)}$	$\emptyset$	$\{emp(0), worksFor(0,0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$
15:A2	[]	$\emptyset$	$\{emp(0), worksFor(0,0)\}$	$\{\leftarrow \underline{worksFor(Y,0)} \wedge \neg boss(Y)\}$	$\{0\}$

Fig. 1.

The transition between two consecutive CQC-nodes, i.e. between an ancestor node and its successor, is a CQC-step that is performed by applying a CQC-expansion rule to a selected literal of the ancestor CQC-node. The selection of literals in the CQC-derivation of Fig. 1 is nearly arbitrary: the only necessary criterion is to avoid picking a non-ground negative-ordinary or built-in literal. In Fig. 1, the CQC-steps are labeled with the name of the CQC-expansion rule that is applied and the selected literal in each step is underlined. We refer to Section 4.2 for a proper formalization of the CQC-expansion rules.

The first step unfolds the selected literal, the IDB atom *sub*$_1$(*X*) from the goal part, by substituting it with the body of its defining rule. At the second step, the selected literal from the goal part is *emp*(*X*), which is a positive EDB literal. To get a successful derivation, i.e. to obtain an EDB satisfying the initial goal, *emp*(*X*) must be true on the constructed EDB. Hence, the method instantiates *X* with a constant and includes the new ground EDB fact in the EDB under construction. The procedure assigns an arbitrary constant to *X*, e.g. 0. So *emp*(0) is the first fact included in the EDB under construction.

¬*chief*(0) is the selected literal in step 3. To get success for the derivation, *chief*(0) must not be true on the EDB. This is guaranteed by adding ← *chief*(0) as a new condition to be enforced. Step 4 is similar to step 3, yielding ← *sub*$_2$(0) to be considered as another condition to be enforced. After performing this later step, we get a CQC-node with a goal like []. However, the work is not done yet, since we must ensure that the two conditions ← *sub*$_2$(0) and ← *chief*(0) are not violated by the current EDB. In other words, we must make both *chief*(0) and *sub*$_2$(0) false.

Step 5 unfolds the selected literal *chief*(0) from one of the two conditions, getting ← *worksFor*(*Y*, 0) ∧ ¬*boss*(*Y*) as a new condition that replaces ← *chief*(0). At least one of the two literals of this condition must be false. In step 6, the selected literal is the positive EDB literal is *worksFor*(*Y*, 0). Since it matches with no EDB atom in the EDB under construction, *worksFor*(*Y*, 0) is false and, consequently, the whole condition ← *worksFor*(*Y*, 0) ∧ ¬*boss*(*Y*) is not violated by the current EDB. For this reason, such a condition is moved from the set of conditions to enforce to the set of conditions to maintain.

Step 7 unfolds the selected IDB atom *sub*$_2$(0) from the remaining condition to enforce. The EDB atom *emp*(0) is the selected literal in step 8. Since *emp*(0) is also present in the EDB under construction, it cannot be false. So this literal is dropped from the condition because it does not help to enforce the condition. In step 9 the selected literal is the negative literal ¬*boss*(0). Since it is the only literal of the condition, it must be made false necessarily. So *boss*(0) becomes a new (sub)goal to achieve and is transferred, thus, to the goal part.

Step 10 unfolds the selected literal *boss*(0) from the goal part as in step 1. *worksFor*(*z*, 0) is the selected literal in step 11. As in step 2, the method should instantiate *Z* with a constant. In this case, the chosen constant is 0 again, so *worksFor*(0, 0) is added to the EDB under construction. Moreover, the condition ← *worksFor*(*Y*, 0) ∧ ¬*boss*(*Y*) is moved back to the set of conditions to enforce to avoid that the new inclusion of *worksFor*(0, 0) in the EDB violates it.

In step 12, the selected literal is the positive EDB literal is *worksFor*(*Y*, 0) from the remaining condition to enforce. Now, it matches with the current contents of the EDB with *Y* = 0. As in step 8, such a literal is dropped from the condition. However, the whole condition ← *worksFor*(*Y*, 0) ∧ ¬*boss*(*Y*) is moved again to the set of conditions to maintain in order to prevent further inclusions of new facts about *worksFor* in the EDB from violating it.

Steps 13 and 14 are identical to steps 9 and 10. In step 15, the constant 0 is selected again to instantiate *worksFor*(*Z*, 0). Since *worksFor*(0, 0) is already included in the

EDB, there is no need to transfer back any condition from the set of conditions to maintain to the set of conditions to enforce.

The CQC-derivation ends successfully since it reaches a CQC-node where the goal to attain is [] and the set of conditions to satisfy is empty. In other words, we can be sure that its EDB, {*emp*(0), *worksFor*(0, 0)}, contains a set of facts that makes the database satisfy the goals and conditions of all preceding CQC-nodes, including, naturally, the first CQC-node. Then we conclude $Q_1 \subset Q_2$.

3.2 Variable Instantiation Patterns

When a CQC-derivation terminates successfully, we obtain a proof, the constructed EDB, which shows that the containment relationship is not true. On the contrary, when a derivation ends unsuccessfully, that is, it terminates but it fails to construct a counterexample, we cannot conclude that containment holds based on a single result. Then the question is how many derivations must be considered before achieving a reliable conclusion. Indeed, rather than the account of all possible derivations, the real point is to know how many variable instantiation alternatives must be considered when adding new facts to the EDB under construction.

The aim of the CQC method is to test only the variable instantiations that are *relevant* without losing completeness. The "strategy" for instantiating the EDB facts to be included in the EDB under construction is connected to, indeed it is inspired by, the concept of *canonical databases* found in [Klu88, Ull97].

Since the canonical databases to be taken into account depend on the concrete subclass of queries that are considered, we distinguish three different *variable instantiation patterns*, VIPs for shorthand. Each of them defines how the CQC method has to instantiate the EDB facts to be added to the EDB under construction. The four VIPs that we define are: *Simple VIP*, *Negation VIP* (as considered in [FTU99]), *Dense Order VIP* and *Discrete Order VIP*.

The CQC method uses the *Simple VIP* when checking containment but not QC under constraints. Moreover, the deductive rules defining query predicates as well as IDB predicates must satisfy the following conditions: they must not have any negative or built-in literal in their rule bodies; they must not have constants in their heads; and they must not have any variable appearing twice or more times in their heads. According to this VIP, each distinct variable is bound to a distinct new constant.

The CQC method uses the *Negation VIP* when checking QCuC or when checking containment under the presence of negated IDB subgoals, negated EDB subgoals and/or (in)equality comparisons ($=, \neq$). In any case, order comparisons ($<, \leq, >, \geq$) are not allowed. EDBs generated and tested with this VIP correspond to the canonical EDBs considered in [Ull97] for the conjunctive query case with negated EDB subgoals. The intuition behind this VIP is clear: Each new variable appearing in a EDB fact to be grounded is instantiated with either some constant previously used or a constant never used before. This is the pattern used in t Fig. 1.

The *Dense Order VIP* and *Discrete Order VIP* are applied when there are order comparisons ($<, \leq, >, \geq$) in the deductive rules, with or without negation. In this case, each distinct variable must be bound to a constant according to either a former or a new location in the total linear order of constants introduced previously [Klu88, Ull97]. The election between these two VIPS depends on whether the comparisons are interpreted on a dense (real numbers) or a discrete order (integer numbers).

4 Formalization of the CQC Method

Let Q_1 and Q_2 be two queries, *DR* the set of deductive rules defining the database IDB relations and *IC* a finite set of conditions expressing the database integrity constraints. If the CQC method performs a successful CQC-derivation from $(\leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)\ \varnothing\ \varnothing\ \varnothing\ K)$ to $([]\ \varnothing\ T\ C\ K')$ then Q_1 c Q_2, where K is the set of constants appearing in $DR \cup Q_1 \cup Q_2$. Moreover, if the CQC method performs a successful CQC-derivation from $(\leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)\ IC\ \varnothing\ \varnothing\ K'')$ to $([]\ \varnothing\ T\ C\ K''')$ then Q_1 c$_{IC}$ Q_2, where K'' is the set of constants appearing in $DR \cup Q_1 \cup Q_2 \cup IC$.

CQC-derivations start from a 5-tuple $(G_0\ F_0\ T_0\ C_0\ K_0)$ consisting of the goal $G_0 = \leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)$, the set of conditions to enforce $F_0 = \varnothing$ or IC, the initially-empty EDB $T_0 = \varnothing$, the empty set of conditions to maintain $C_0 = \varnothing$ and the set K_0 of constant values appearing in $DR \cup Q_1 \cup Q_2[\cup IC]$.

A successful CQC-derivation reaches a 5-tuple $(G_n\ F_n\ T_n\ C_n\ K_n) = ([]\ \varnothing\ T\ C\ K')$, where the empty goal $G_n = []$ means that we have reached the goal G_0 we were looking for. The empty set $F_n = \varnothing$ means that no condition is waiting to be satisfied. $T_n = T$ is an EDB that satisfies G_0 as well as F_0. $C_n = C$ is a set of conditions recorded along the derivation and that T also satisfies. $K_n = K'$ is the set of constant values appearing in $DR \cup Q_1 \cup Q_2[\cup IC] \cup T$.

On the contrary, if every "fair" CQC-derivation starting from $(\leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)\ \varnothing\ [\cup IC]\ \varnothing\ \varnothing\ K)$ is finite but does not reach $([]\ \varnothing\ T\ C\ K')$, it will mean that no EDB satisfies the goal $G_0 = \leftarrow q_1(X_1, ..., X_n) \wedge \neg q_2(X_1, ..., X_n)$ together with the set of conditions $F_0 = \varnothing\ [\cup IC]$, concluding that Q_1 S Q_2 (Q_1 S$_{IC}$ Q_2). Section 5 below provides the complete results and proofs regarding the soundness and completeness of the CQC method.

4.1 CQC-Nodes, CQC-Trees and CQC-Derivations

Let Q_1 and Q_2 be two queries, *DR* be the set of deductive rules defining the database IDB relations and *IC* be a finite set of conditions expressing the database integrity constraints. A *CQC-node* is a 5-tuple of the form $(G_i\ F_i\ T_i\ C_i\ K_i)$, where G_i is a goal to attain; F_i is a set of conditions to enforce; T_i is a set of ground EDB atoms, an EDB under construction; C_i is a set of conditions that are currently satisfied in T_i and must be maintained; and K_i is the set of constants appearing in $R = DR \cup Q_1 \cup Q_2[\cup IC]$ and T_i.

A *CQC-tree* is inductively defined as follows:

1. The tree consisting of the single CQC-node $(G_0\ F_0\ \varnothing\ \varnothing\ K)$ is a CQC-tree.
2. Let E be a CQC-tree, and $(G_n\ F_n\ T_n\ C_n\ K_n)$ a leaf CQC-node of E such that $G_n \neq []$ or $F_n \neq \varnothing$. Then the tree obtained from E by appending one or more descendant CQC-nodes according to a CQC-expansion rule applicable to $(G_n\ F_n\ T_n\ C_n\ K_n)$ is again a CQC-tree.

It may happen that the application of a CQC-expansion rule on a leaf CQC-node (G_n F_n T_n C_n K_n) does not obtain any new descendant CQC-node to be appended to the CQC-tree because some necessary constraint defined on the CQC-expansion rule is not satisfied. In such a case, we say that (G_n F_n T_n C_n K_n) is a *failed* CQC-node.

Each branch in a CQC-tree is a *CQC-derivation* consisting of a (finite or infinite) sequence (G_0 F_0 T_0 C_0 K_0), (G_1 F_1 T_1 C_1 K_1), … of CQC-nodes.

A CQC-derivation is *finite* if it consists of a finite sequence of CQC-nodes; otherwise it is *infinite*. A CQC-derivation is *successful* if it is finite and its last (leaf) CQC-node has the form ([] $\varnothing$ T_n C_n K_n). That is, both the goal to attain and the set of conditions to satisfy are empty. A CQC-derivation is *failed* if it is finite and its last (leaf) CQC-node is failed.

A CQC-tree is *successful* when at least one of its branches is a successful CQC-derivation. A CQC-tree is *finitely failed* when each one of its branches is a failed CQC-derivation.

4.2 The CQC-Expansion Rules

The nine *CQC-expansion rules* are listed in tables 4.1 and 4.2. For the sake of notation, if $G_i = \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_j \wedge L_{j+1} \wedge \ldots \wedge L_m$ then $G_i \backslash L_j = \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_m$. If $G_i = \leftarrow L_1 \wedge \ldots \wedge L_m$ then $G_i \wedge p(\bar{X}) = \leftarrow L_1 \wedge \ldots \wedge L_m \wedge p(\bar{X})$.

Table 1. CQC-expansion rules: A#-rules.

A1) $P(G_i) = d(\bar{X})$ is a positive IDB atom:

$$\frac{(G_i\ F_i\ T_i\ C_i\ K_i)}{(G_{i+1,1}\ F_i\ T_i\ C_i\ K_i) \mid \ldots \mid (G_{i+1,m}\ F_i\ T_i\ C_i\ K_i)}$$

only if m ≥ 1 and each $G_{i+1,j}$ is the resolvent for G_i and some deductive rule $d(\bar{Y}) \leftarrow M_1 \wedge \ldots \wedge M_q$ in R.

A2) $P(G_i) = b(\bar{X})$ is a positive EDB atom:

$$\frac{(G_i\ F_i\ T_i\ C_i\ K_i)}{((G_i \backslash b(\bar{X}))\sigma_1\ F_{i+1,1}\ T_{i+1,1}\ C_{i+1,1}\ K_{i+1,1}) \mid \ldots \mid ((G_i \backslash b(\bar{X}))\sigma_m\ F_{i+1,m}\ T_{i+1,j}\ C_{i+1,m}\ K_{i+1,m})}$$

such that $F_{i+1,j} = F_i \cup C_i$, $T_{i+1,j} = T_i \cup \{b(\bar{X})\sigma_j\}$ and $C_{i+1,j} = \varnothing$ if $b(\bar{X})\sigma_j \notin T_i$; otherwise $F_{i+1,j} = F_i$, $T_{i+1,j} = T_i$ and $C_{i+1,j} = C_i$. Each σ_j is one out of m possible distinct ground substitutions, obtained via a variable instantiation procedure from ($vars(\bar{X})$, $\varnothing$, K_i) to ($\varnothing$, σ_j, $K_{i+1,j}$) according to the appropriate variable instantiation pattern, that assigns a constant from $K_{i+1,j}$ to each variable in $vars(\bar{X})$. See more details in [FTU02].

A3) $P(G_i) = \neg p(\bar{X})$ is a ground negated atom:

$$\frac{(G_i\ F_i\ T_i\ C_i\ K_i)}{(G_i \backslash \neg p(\bar{X})\ F_i \cup \{\leftarrow p(\bar{X})\}\ T_i\ C_i\ K_i)}$$

A4) $P(G_i)$ =L is a ground built-in literal:

$$\frac{(G_i\ F_i\ T_i\ C_i\ K_i)}{(G_i \backslash L\ F_i\ T_i\ C_i\ K_i)}$$

only if L is evaluated true.

Table 2. CQC-expansion rules: B#-rules.

B1) $P(F_{i,j}) = d(\bar{X})$ is a positive IDB atom:

$$\frac{(G_i \; \{F_{i,j}\} \cup F_i \; T_i \; C_i \; K_i)}{(G_i \; S \cup F_i \; T_i \; C_i \; K_i)}$$

where S is the set of all resolvents S_u for clauses in R and $F_{i,j}$ on $d(\bar{X})$. S may be empty.

B2) $P(F_{i,j}) = b(\bar{X})$ is a positive EDB atom:

$$\frac{(G_i \; \{F_{i,j}\} \cup F_i \; T_i \; C_i \; K_i)}{(G_i \; S \cup F_i \; T_i \; C_{i+1} \; K_i)}$$

only if $[\,] \notin S$.
$C_{i+1} = C_i$ if $\bar{X}$ contains no variables and $b(\bar{X}) \in T_i$; otherwise, $C_{i+1} = C_i \cup \{F_{i,j}\}$
S is the set of all resolvents of clauses in T_i with $F_{i,j}$ on $b(\bar{X})$. S may be empty, meaning that $b(\bar{X})$ cannot be unified with any atom in T_i.

B3) $P(F_{i,j}) = \neg p(\bar{X})$ is a ground negative ordinary literal:

$$\frac{(G_i \; \{F_{i,j}\} \cup F_i \; T_i \; C_i \; K_i)}{(G_i \; \{\leftarrow p(\bar{X})\} \cup \{F_{i,j} \backslash \neg p(\bar{X})\} \cup F_i \; T_i \; C_i \; K_i) \text{ only if } F_{i,j} \backslash \neg p(\bar{X}) \neq [\,] \quad | \quad (G_i \wedge p(\bar{X}) \; F_i \; T_i \; C_i \; K_i)}$$

B4) $P(F_{i,j}) = L$ is a ground built-in literal that is evaluated true:

$$\frac{(G_i \; \{F_{i,j}\} \cup F_i \; T_i \; C_i \; K_i)}{(G_i \; \{F_{i,j} \backslash L\} \cup F_i \; T_i \; C_i \; K_i)}$$

only if $F_i \backslash L \neq [\,]$.

B5) $P(F_{i,j}) = L$ is a ground built-in literal that is evaluated false:

$$\frac{(G_i \; \{F_{i,j}\} \cup F_i \; T_i \; C_i \; K_i)}{(G_i \; F_i \; T_i \; C_i \; K_i)}$$

Once a literal is selected, only one of the CQC-expansion rules can be applied. We distinguish two classes of rules: A-rules and B-rules. A-rules are those where the selected literal belongs to the goal G_i. Instead, B-rules correspond to those where the selected literal belongs to any of the conditions $F_{i,j}$ in F_i. Inside each class of rules, they are differentiated with respect to the type of the selected literal.

In each CQC-expansion rule, the part above the horizontal line presents the CQC-node to which the rule is applied. Below the horizontal line is the description of the resulting descendant CQC-nodes. Vertical bars separate alternatives corresponding to different descendants. Some rules like A1, A5, B2 and B4 include also an "only if" condition that constraints the circumstances under which the expansion is possible. If such a condition is evaluated false, the CQC-node to which the rule is applied becomes a failed CQC-node.

Finally, note that three CQC-expansion rules, namely A1, B1 and B2, use the resolution principle as is defined in [Llo87].

The application of a CQC-expansion rule on a given CQC-node (G_i F_i T_i C_i K_i) may result in none, one or several alternative (branching) descendant CQC-nodes depending on the selected literal $P(J_i) = L$. Here, J_i is either the goal G_i or any of the conditions $F_{i,j}$ in F_i. L is selected according to a safe computation rule P [Llo87], which selects negative and built-in literals only when they are fully grounded. To guarantee that such literals are sooner or later selected we require deductive rules and goals to be safe.

5 Correctness Results for the CQC Method

In this Section, we summarize and sketch the new proofs of correctness of the CQC method. We refer the reader to [FTU02] for the detailed proofs. We also state the class of queries that can be actually decided by the CQC method. Before proving these results, we need to make explicit the model-theoretic semantics to with respect those results are established.

Let R be a set of deductive rules. We define the *partial completion* of R, denoted by $pComp(R)$, as the collection of completed definitions [Cla77] of IDB predicates in R together with an equality theory. This later one includes a set of axioms stating explicitly the meaning of the predicate "=" introduced in the completed definitions.

Our partial completion is defined similarly to Clark's completion [Cla77], $Comp(R)$, but without including the axioms of the form $\forall x(\neg b_i(\bar{X}))$ for each predicate b_i which only occurs in the body of the clauses in R. We assume that these predicates are EDB predicates that, obviously, are not defined in R.

If Q_1 and Q_2 are two queries, DR is the set of deductive rules defining the database IDB relations and IC be a finite set of conditions expressing the database integrity constraints, we consider that problem of knowing whether Q_1 S Q_2 *(*Q_1 S$_{IC}$ Q_2*)* is equivalent to the problem of proving that $pComp(R)[\cup \forall IC] \setminus \forall X_1 \ldots X_n \; q_1(X_1,\ldots,X_n) \rightarrow q_2(X_1,\ldots,X_n)$ is true, where $R = DR \cup Q_1 \cup Q_2$. If we define the initial goal $G_0 = \leftarrow q_1(X_1,\ldots,X_n) \wedge \neg q_2(X_1,\ldots,X_n)$ then testing Q_1 S Q_2 *(*Q_1 S$_{IC}$ Q_2*)* is equivalent to proving $pComp(R)[\cup\forall IC] \setminus G_0$. This proof is tackled by the CQC method, which tries to refute $pComp(R)[\cup\forall IC] \setminus G_0$ by constructing an EDB T such that $R(T)$ is a model for $pComp(R)\,[\cup\forall IC] \cup \{\exists X_1 \ldots \exists X_n\,(q_1(X_1,\ldots,X_n) \wedge \neg q_2(X_1,\ldots,X_n))\}$.

In the following theorems, let $G_0 = \leftarrow q_1(X_1, \ldots, X_n) \wedge \neg q_2(X_1, \ldots, X_n)$ be the initial goal, $F_0 = \emptyset\ [\cup IC]$ be the initial set of conditions to enforce and K be the set of constants appearing in $DR \cup Q_1 \cup Q_2 \cup F_0$.

Before proving results related to failure of the CQC method, we review the results related to finite success already stated in [FTU99].

Theorem 5.1 (*Finite Success Soundness*)

If there exists a finite successful CQC-derivation starting from $(G_0\ F_0\ \emptyset\ \emptyset\ K)$ then Q_1 c Q_2 (Q_1 c$_{IC}$ Q_2) provided that $\{G_0\} \cup F_0 \cup DR \cup Q_1 \cup Q_2$ is safe and hierarchical.

Theorem 5.2 (*Finite Success Completeness*)

If Q_1 c Q_2 *(*or Q_1 c$_{IC}$ Q_2*)* then there exists a successful CQC-derivation from $(G_0\ F_0\ \emptyset\ \emptyset\ K)$ to $([]\ \emptyset\ T\ C\ K')$ provided that $\{G_0\} \cup F_0 \cup DR \cup Q_1 \cup Q_2$ is safe and either hierarchical or strict-stratified [CL89].

These results ensure that, in the absence of recursive IDB predicates, if the method builds a finite counterexample, then containment does not hold (Theorem 5.1); and that if there exists a finite counterexample, then our method finds it and terminates (Theorem 5.2). We extend these results by assessing the properties regarding failure of our method. In this sense, we prove *failure soundness* (Theorem 5.3), which guarantees that if the method terminates without building any counterexample then containment holds; and *failure completeness* (Theorem 5.5), which states that if containment holds between two queries then our method fails finitely.

Theorem 5.3 (*Failure Soundness*)

If there exists a finitely failed CQC-Tree rooted at $(G_0\ F_0\ \varnothing\ \varnothing\ K)$ then Q_1 S Q_2 (Q_1 S$_{\text{IC}}$ Q_2) provided that the deductive rules and conditions in $DR \cup Q_1 \cup Q_2[\cup IC]$ are safe.

The proof of Theorem 5.3 is made by using the principle of contradiction and may be intuitively explained as follows. Le us suppose that we have a finitely failed CQC-tree but Q_1 c Q_2 (Q_1 c$_{\text{IC}}$ Q_2). If Q_1 c Q_2 (Q_1 c$_{\text{IC}}$ Q_2) it means for us that $pComp(R)$ $[\cup\forall IC] \cup \{\exists X_1...\exists X_n\ (q_1(X_1,\ldots,X_n) \wedge \neg q_2(X_1,\ldots,X_n))\}$ has a model. However, if this is true, we prove that there is at least one CQC-derivation not finitely failed.

Lemma 5.4 is needed for proving Theorem 5.5. Before stating it, we need some new definitions.

A CQC-derivation is *open* when it is not failed. That is, when the derivation is either infinite or finite with its last (leaf) CQC-node having the form of $([]\ \varnothing\ T_n\ C_n\ K_n)$. A CQC-derivation θ is *saturated for the CQC-expansion Rules* if for every CQC-node $(G_i\ F_i\ T_i\ C_i\ K_i)$ in θ the following properties hold:

1. For each literal $L_{i,j} \in G_i$ there exists a node $(G_n\ F_n\ T_n\ C_n\ K_n)$, $n \geq i$, such that $P(G_n) = L_{i,j}\sigma_{i+1}\ldots\sigma_n$ is the selected literal on that node to apply a CQC A-rule, where $\sigma_{i+1}\ldots\sigma_n$ is the composition of the substitutions used in the intermediate nodes.
2. For each condition $F_{i,j_i} \in F_i$ there exists a node $(G_n\ F_n\ T_n\ C_n\ K_n)$, $n \geq i$, such that $F_{n,j_n} \in F_n$ is the selected condition on that node to apply a CQC B-rule and $F_{n,j_n} = F_{i,j_i}$.

A CQC-derivation is said to be *fair* when it is either failed or open and saturated for the CQC-expansion Rules. A CQC-tree is *fair* if each one of its CQC-derivations (branches) is fair. Note that a finitely failed CQC-tree is always fair, but the inverse is not necessarily true.

Lemma 5.4

Let R be a set of deductive rules, $G = \leftarrow L_1 \wedge \ldots \wedge L_k$ be a goal, F be a set of conditions and K be the set of constants in $\{G_0\} \cup F_0 \cup R$. If there exists a saturated open CQC-derivation starting from $(G_0\ F_0 \varnothing\ \varnothing\ K)$ then $pComp(R) \cup \{\exists(L_1 \wedge \ldots \wedge L_k)\} \cup \forall F_0$ has a model provided that $\{G_0\} \cup F_0 \cup R$ is safe and hierarchical.

Theorem 5.5 (*Failure Completeness*)

If Q_1 S Q_2 *(*Q_1 S$_{\text{IC}}$ Q_2*)* then every fair CQC-Tree rooted at $(G_0\ F_0\ \varnothing\ \varnothing\ K)$ is finitely failed provided that $\{G_0\} \cup F_0 \cup DR \cup Q_1 \cup Q_2$ is safe and hierarchical.

The proof is made by contradiction and may be intuitively explained as follows. If Q_1 S Q_2 (Q_1 S$_{\text{IC}}$ Q_2) then $pComp(R)$ $[\cup\forall IC] \cup \{\exists X_1\ldots X_n\quad q_1(X_1,\ldots,X_n) \wedge$

$\neg q_2(X_1,\ldots,X_n)\}$ cannot have a model. Assuming that it is true, let us suppose that we have a non-failed CQC-derivation starting from $(G_0\ F_0\ \varnothing\ \varnothing\ K)$. However, lemma 5.4 shows that this derivation would indeed construct a model for $pComp(DR)$ $[\cup\forall IC]$ $\cup$ $\{\exists X_1 ... \exists X_n\ (q_1(X_1,\ldots,X_n) \wedge \neg q_2(X_1,\ldots,X_n))\}$.

6 Decidability Results

The QC problem is undecidable for the general case of queries and databases that the CQC Method covers [AHV95]. A possible source of undecidability is the presence of recursion, which could make the CQC Method build and test an infinite number of EDBs. In this sense, the CQC Method excludes explicitly the presence of any type of recursion as we have seen in the proofs of the failure completeness (Theorem 5.5) and the finite success soundness and completeness (Theorems 5.1 and 5.2).

Another reason for undecidability is the presence of “axioms of infinity” [BM86]. In this case, the initial goal to attain could only be satisfied on an EDB with an infinite number of facts because each new addition of a fact to the EDB under construction triggers a condition to be *repaired* with another insertion on the EDB.

For this reason, the CQC Method is semidecidable for the general case, in the sense that if either there exist one or more finite EDBs for which containment does not hold or there is no EDB (finite or infinite), the CQC Method terminates according to our completeness results (Theorems 5.2 and 5.5). Nevertheless, we cannot guarantee termination under the presence of infinite counterexamples.

One of the forms to assure always termination when using the CQC Method is to delimit a priori the type of schemas and queries for which it is guaranteed that infinite non-containment counterexamples never exist. It is well known that this is the case of all the different classes of conjunctive queries, including those allowing negated EDB atoms and built-in atoms. Then, we can guarantee that our method will always terminate in these cases.

7 Related Work

As we have seen, the CQC method deals with queries and database schemas that include negation on IDB predicates, integrity constraints and built-in order predicates. Previous methods that deal with negated IDB subgoals can be classified in two different approaches: either they check Uniform Query Containment or they consider just restricted cases of negation.

[LS93] checks *Uniform QC* for queries and databases with safe negated IDB atoms. The problem is that, as pointed out in [Sag88], Uniform QC (written $Q_1\ \mathrm{S}^u\ Q_2$) is a sufficient but not necessary condition for query containment. That is, if for a given pair of queries Q_1 and Q_2 we have that $Q_1\ \mathrm{S}^u\ Q_2$, then $Q_1\ \mathrm{S}\ Q_2$. On the other hand, if the result is that $Q_1\ \mathrm{c}^u\ Q_2$, then nothing can be said about whether or not $Q_1\ \mathrm{S}$ Q_2 holds. In contrast, we have seen that the CQC method always checks “true” query containment.

The rest of the methods that handle negation restrict the classes of queries and database schemas they are able to deal with. Thus, we have that [Ull97, WL03] consider only conjunctive queries with negated EDB predicates, while [LS95] checks containment of a datalog query in a conjunctive query with negated EDB predicates. [CDL98] cannot express simple cases of negation on IDB predicates since it is not possible to define negation in the regular expression they consider.

In the long version of this paper [FTU03], we show the clear correspondence between the CQC Method and the algorithms defined in [Ull97] and [WL03] by means of examples. Our conclusion is that the CQC Method is not less efficient than those two outstanding methods for the cases that they handle.

When checking containment for conjunctive queries with negated EDB predicates, both the CQC Method with the Negation VIP and the algorithm of [Ull97] achieve the same results, but their strategies are different. The CQC builds and tests canonical EDBs dynamically since it finds one that fulfils the initial goal to attain or since no canonical EDB, with or without extension, satisfies the goal after having built all. Instead, the method of [Ull97] first builds all the canonical EDBs and then, it tests if each of them accomplishes the containment relationship.

The algorithm of [Ull97] can be easily extended to consider order predicates in the rule bodies of conjunctive queries over the two types of interpretations, dense or discrete. In this case, the canonical databases that would be built it should take into account every possible total ordering of variables to instantiate. Again, the CQC Method not only covers this class of queries but also constructs similar (canonical) EDBs with the Dense Order or the Discrete Order VIPs.

The algorithm proposed in [WL03] to check conjunctive query containment with safe negated EDB atoms improves the efficiency of [Ull97] since it does not generate necessarily the complete set of canonical EDBs that the method of [Ull97] needs to construct. If we adopt the theoretical results in which the algorithm of [WL03] is based, then it follows that the Simple VIP may replace the Negation VIP when using the CQC Method to check query containment for conjunctive queries with negated EDB subgoals, without any loss of completeness. In this way, the CQC Method with the Simple VIP and the algorithm of [WL03] become quite similar. Therefore, we do not need to generate all the canonical EDBs that the CQC Method with the Negation VIP and [Ull97] would consider and, accordingly, the CQC Method with the Simple VIP is as efficient as the algorithm in [WL03] for the cases covered by this latter.

8 Conclusions

In this paper we have presented the Constructive Query Containment (CQC) method for QC Checking which ckecks "true" QC and QCuC for queries over databases with safe negation in both IDB and EDB subgoals and with or without built-in predicates. As far as we know, ours is the first proposal that covers all these features in a single method and in a uniform and integrated way.

We have proved several properties regarding the correctness of the CQC method: finite success soundness for hierarchical queries and databases, failure soundness, finite success completeness for strict-stratified queries and databases and failure completeness for hierarchical queries and databases. From these results, and from previous results that showed that infinite non-containment counterexamples never exist in the particular case of checking QC for conjunctive queries with safe EDB negation and built-in predicates, we can ensure termination, and thus decidability, of our method for those cases.

The main contributions of this paper are twofold. First, we have shown that the CQC method performs containment tests for more and broader cases of queries and database schemas than previous methods. Second, we have also shown that the CQC method is decidable and not less efficient than other methods to check query containment of conjunctive queries with or without safe negated EDB predicates.

As a further work, we plan to characterize other classes of queries and deductive rules for which our method always terminates.

References

[AHV95] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases. Addison Wesley, 1995.

[BM86] F. Bry, R. Manthey. Checking Consistency of Database Constraints: a Logical Basis. In Proceedings of VLDB'86, 13–20, 1986.

[CDL98] D. Calvanese, G. De Giacomo, M. Lenzerini. On the Decidability of Query Containment under Constraints. In Proceedings of the PODS'98, 149–158, 1998.

[CL89] L. Cavedon, J.W. Lloyd. A Completeness Theorem for SLDNF Resolution, Journal of Logic Programming, 7(3):177–191, 1989.

[Cla77] K.L. Clark. Negation as Failure. In Logic and Data Bases, 293–322, Plenum Press, 1977

[CM77] A.K. Chandra, P.M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In Proc. of the 9th ACM SIGACT Symposium on Theory of Computing, 77–90, 1977.

[FTU99] C. Farré, E. Teniente, T. Urpí. The Constructive Method for Query Containment Checking. In Proceedings of the DEXA'99, 583–593, 1999.

[FTU02] C. Farré, E. Teniente, T. Urpí. Formalization And Correctness Of The CQC Method. Technical Report LSI-02–68-R.

[FTU03] C. Farré, E. Teniente, T. Urpí. Query Containment With Negated IDB Predicates (Extended Version). Technical Report LSI-03–22-R.

[Klu88] A. Klug. On Conjunctive Queries Containing Inequalities. Journal of the ACM, 35(1):146–160, 1988.

[Llo87] J.W. Lloyd. Foundations of Logic Programming, Springer, 1987.

[LR96] A. Levy, M-C. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In Proc. of the ECAI'96, 323–327, 1996.

[LS93] A. Levy, Y. Sagiv. Queries Independent of Updates. In Proceedings of the VLDB'93, 171–181, 1993.

[LS95] A. Levy, Y. Sagiv. Semantic Query Optimization in Datalog Programs. In Proceedings of PoDS'95, 163–173, 1995.

[Sag88] Y. Sagiv. Optimizing Datalog Programs. In Foundations of Deductive Databases and Logic Programming, 659–698, Morgan Kaufmann, 1988.

[Ull97] J. D. Ullman. Information Integration Using Logical Views. In Proc. of the ICDT'97, 19–40, 1997

[WL03] F. Wei, G. Lausen. Containment of Conjunctive Queries with Safe Negation. In Proceedings of ICDT'03: 346–360, 2003.

Author Index